Министерство образования и науки Украины
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ
«КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ имени ИГОРЯ СІКОРСКОГО»

На правах рукописи

Диброва Михаил Александрович

УДК 004.04 (043.3)

Способ многопутевой маршрутизации в компьютерных сетях большой размерности

Специальность 05.13.05 – Компьютерные системы и компоненты
Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
Кулаков Юрий Алексеевич
доктор технических наук,
профессор

Киев – 2017

ВВЕДЕНИЕ

Суть научной задачи, решаемой в рамках диссертационной работы, состоит в разработке способа многопутевой маршрутизации, ориентированного на компьютерные сети большой размерности. Предложенный в работе способ формирования множества непересекающихся путей между абонентами компьютерной сети позволяет повысить эффективность управления трафиком в компьютерных сетях большой размерности и обеспечить заданные значения параметров качества обслуживания (QoS). Решение данной задачи базируется на методах дискретной математики, теории графов и методах комбинаторного анализа. Использование графо-комбинаторных методов маршрутизации позволяет уменьшить временную сложность процедуры формирования максимально возможного множества непересекающихся путей в компьютерной сети большой размерности.

Актуальность работы. Расширение сферы использования современных сетевых технологий предъявляет новые, более высокие требования к качеству обслуживания (QoS) на уровне систем передачи информации и конструирования трафика. В процессе эксплуатации компьютерных сетей динамически изменяются такие характеристики как пропускная способность каналов передачи данных, загрузка узлов коммутации, реконфигурация сети. Все это приводит к необходимости осуществлять ремаршрутизацию, что в свою очередь увеличивает объем служебного трафика и сказывается на эффективности функционирования компьютерной сети. Одним из наиболее эффективных путей решения этой задачи является использование многопутевой маршрутизации, которая за счет сокращения времени реконфигурации маршрутов обеспечивает равномерную загрузку каналов передачи данных и позволяет повысить эффективность процедуры конструирования трафика.

Таким образом, тематика диссертационной работы, направленная на повышение эффективности функционирования компьютерных сетей за счет разработки и усовершенствования комбинаторных алгоритмов формирова-

ния множества непересекающихся путей, является актуальной и представляет научный и практический интерес.

Связь работы с научными программами, планами, темами

Работа выполнялась в соответствии с планами научных исследований на кафедре вычислительной техники Национального технического университета Украины «Киевский политехнический институт» в рамках научно-исследовательских работ:

– «Разработка теоретических основ построения высокопроизводительных компьютерных систем с динамическим распараллеливанием вычислительных процессов» (государственный регистрационный номер 0111U002729 согласно научным направлениям «Разработка высокопроизводительных многокластерных вычислительных систем»);

– «Методы организации мониторинговых информационно-аналитических систем научно-образовательного назначения на основе высокопроизводительных вычислительных кластерных технологий» (государственный регистрационный номер 0109U000526).

Цель и задачи исследования. Целью диссертационной работы является повышение эффективности функционирования компьютерных сетей большой размерности за счет организации многопутевой маршрутизации на основе способа формирования максимального множества непересекающихся путей с минимальной временной сложностью.

Для достижения поставленной цели в работе решаются такие задания:

1. Анализ протоколов многопутевой маршрутизации с точки зрения их временной сложности.

2. Определение критериев наличия максимального количества непересекающихся путей между двумя узлами сети.

3. Анализ способов формирования множества непересекающихся путей с целью выбора алгоритма многопутевой маршрутизации с минимальной временной сложностью.

4. Разработка алгоритма формирования максимального множества непересекающихся путей между двумя вершинами на основе модифицированного метода «ветвей и границ».

5. Анализ возможности и разработка алгоритма формирования множества непересекающихся путей на основе сочетания методов «ветвей и границ» и «встречной волны».

6. Разработка способа параллельной передачи частей сообщения по множеству непересекающихся путей между двумя узлами компьютерной сети.

7. Разработка метода одновременного формирования множества непересекающихся путей от одного узла к нескольким узлам компьютерной сети.

8. Организация многопутевой маршрутизации в сети MPLS.

9. Разработка способа многопутевого доступа к распределенным центрам данных.

10. Моделирование и анализ эффективности предложенного способа и алгоритмов многопутевой маршрутизации в компьютерных сетях большой размерности.

Объектом исследований является процесс маршрутизации в компьютерных сетях большой размерности.

Предметом исследования являются способы и средства формирования многопутевых виртуальных каналов передачи информации в компьютерных сетях.

Методы исследования основываются на теории множеств и теории графов, методах комбинаторного анализа, а также использовании методов имитационного моделирования.

Научная новизна полученных результатов заключается в новом подходе к решению задачи многопутевой маршрутизации в компьютерных сетях большой размерности.

Автором получены следующие новые научные результаты:

1. Предложены и обоснованы критерии формирования максимально возможного множества непересекающихся путей в глобальных компьютерных сетях.

2. Получил дальнейшее развитие метод «ветвей и границ», который за счет одновременного формирования множества путей от одного узла к нескольким узлам, позволяет за счет исключения повторных операции направленного перебора, возникающих при последовательном формировании путей, существенным образом уменьшить временную сложность формирования структуры виртуальной компьютерной сети.

3. Впервые предложен и обоснован способ формирования частично пересекающихся путей, который за счет формирования непересекающихся путей до вершин области сочленения подграфов позволяет существенным образом уменьшить временную сложность алгоритма формирования непересекающихся и частично пересекающихся путей.

4. Впервые на основе метода сочленения деревьев предложен и обоснован способ формирования множества непересекающихся путей, в качестве корневых вершин которых выступают узлы компьютерной сети, который за счет использования алгоритма «встречной волны» позволяет сократить время и область поиска множества непересекающихся путей.

5. Получил дальнейшее развитие способ формирования стека меток в сети MPLS, отличающийся от известных линейной временной сложностью и позволяющий организовать передачу информации по множеству непересекающихся путей между граничными маршрутизаторами сети MPLS.

6. Предложен и обоснован способ многопутевой маршрутизации в распределенных центрах данных, который за счет учета самоподобия структуры системы доступа к узлам распределенных центров процедуры обхода древовидной виртуальной структуры системы доступа к ресурсам данных, позволяет оптимизировать процедуру формирования и поиска наиболее оптимальных путей передачи информации.

Практическое значение полученных результатов определяется тем, что:

• Предложенный способ организации виртуальных вычислительных сетей с многопутевыми каналами передачи информации за счет организации многопутевой маршрутизации позволяет повысить эффективность функционирования распределенных вычислительных систем большой размерности.

• Разработанные средства формирования многопутевых виртуальных сетей позволяют повысить эффективность процедуры управления трафиком в распределенных вычислительных системах большой размерности и оптимизировать процесс доступа к ресурсам в центрах данных.

Личный вклад соискателя. Основные результаты получены автором самостоятельно. В работах, написанных в соавторстве автором, получены следующие результаты:

1. Способ одновременного формирования множества путей от одного узла к нескольким узлам.

2. Алгоритм формирования структуры grid системы, ориентированной на многопутевую маршрутизацию.

3. Модифицированный алгоритма «обратной волны» формирования множества непересекающихся путей за счет пересечения деревьев, корневыми вершинами которых являются узлы сети, между которыми формируются пути передачи информации.

4. Алгоритм формирования стека меток, характеризующийся линейной временной сложностью и позволяющий организовать передачу информации по множеству непересекающихся между граничными маршрутизаторами сети MPLS.

5. Способ формирования множества путей в сетевых центрах данных.

Апробация результатов диссертации. Результаты исследований диссертации докладывались, обсуждались и получили положительную оценку на:

1. Пятой Международной научно- практической конференции «Методи та засоби кодування, захисту й ущільнення інформації» г. Винница, 19-21 квітня 2016 року. – Винница: ВНТУ, 2016.

2. XVI международной научной конференции «Интеллектуальный анализ информации ИАИ-2016» им. Таран Т.А. -Киев: Просвіта – 2016.

Публикации. По результатам выполненных исследований опубликовано 8 научных работ, из которых 6 статей в профессиональных научно-технических специализированных изданиях, которые реферируются научно метрическими базами, 2 публикации в сборниках трудов Международных научно-технических конференций.

ГЛАВА 1

АНАЛИЗ СПОСОБОВ МНОГОПУТЕВОЙ МАРШРУТИЗАЦИИ В КОМПЬЮТЕРНЫХ СЕТЯХ БОЛЬШОЙ РАЗМЕРНОСТИ

1.1. Обзор способов конструирования трафика в системах распределенной обработки и хранения информации на основе многопутевой маршрутизации

В настоящее время в распределенных системах обработки информации, таких как Grid и Cloud системы, широко используется технология виртуальных вычислительных сетей. Данная технология позволяет уменьшить временную сложность процедуры маршрутизации на этапе формирования путей передачи информации. В свою очередь организация многопутевой маршрутизации позволяет:

- повысить эффективность доступа к распределенным ресурсам за счет оптимизации процедуры конструирования трафика (TE);
- обеспечить более равномерную загрузку сети;
- организовать параллельную передачу информации;
- повысить безопасность и надежность передачи данных.

Одним из путей развития технологии виртуальных вычислительных сетей является организация многопутевых виртуальных каналов, каждый из которых включает в свой состав несколько путей передачи информации. Организация многопутевой маршрутизации в рамках виртуальных каналов позволяет упростить и повысить эффективность процедуры управления трафиком в распределенных системах обработки информации. Использование многопутевых виртуальных каналов позволяет организовать распределенную многопутевую маршрутизацию, так в работе [1] предложен алгоритм распределенного управления трафиком при многопутевой маршрутизации, обеспечивающий оптимизацию нагрузки на сеть передачи и минимизирующий стоимость передачи.

В работе [2] предложен способ конструирования трафика на основе адаптивной многопутевой маршрутизации при использовании виртуальных каналов.

В работе [3] предложен способ многопутевой маршрутизации, обеспечивающий живучесть и надежность передачи информации на основе многопутевой маршрутизации. При этом отмечается актуальность разработки множества непересекающихся путей с достаточной пропускной способностью.

В настоящее время многопутевая маршрутизация широко используется для организации безопасной маршрутизации в мобильных компьютерных сетях [4,5,6,7,8,9,10,11]. Организация процесса нахождения множества путей определена с помощью модифицированного алгоритма Дейкстры, который используется в протоколах маршрутизации. В связи с значительной временной сложностью данного алгоритма при использовании его для многопутевой маршрутизации возникает необходимость его модификации [12].

Преимущество распределенных алгоритмов управления трафиком в том, что они настраиваются в одном временном масштабе (порядка RTTs), и способны быстро реагировать на изменения трафика [13].

В табл. 1.1 приведен сравнительный обзор распределенных протоколов управления трафиком на основе многопутевой маршрутизации [14]:

Таблица 1.1

Сравнение распределенных протоколов маршрутизации

Технология	Описание	Распределенный	Контроль перегрузки	Управление трафиком
TRUMP	По состоянию каналов изменяет скорость источника	да	да	нет
PEFT	маршрутизация по состоянию каналов	да	нет	да
DEFT	маршрутизация по состоянию каналов	да	да	да
DATE	минимизирует нагрузку на сеть	да	да	да

TeXCP	Балансирует нагрузку на основании обратной связи.	да	нет	да
MATE	Сводит к минимуму сумму задержек в сети	Частично распределенный	нет	да

Для реализации распределенного алгоритма несколько путей должны существовать между парой источник-назначение, источники должны знать и контролировать несколько путей, и переадресация пакетов должна быть возможной на каждом из указанных путей. В алгоритме можно сопоставить “источники” и “пути” с разными элементами сети в зависимости от сценария. Например, в современной системе управления трафиком источники могут быть конечными узлами (использующими распределенные алгоритмы управления перегрузками), или граничными маршрутизаторами (предложенными для запуска распределенного трафика протоколов проектирования).

Множественные пути: из-за того, что множество мелких сетей подключены к нескольким высокоуровневым провайдерам, часто существуют множественные end-to-end пути, большинство Интернет-провайдеров имеют несколько путей между парой граничных маршрутизаторов, а крупные провайдеры часто подключаются друг к другу в нескольких точках. Большинство протоколы маршрутизации пересылают пакеты только на одном пути между источником и назначением, хотя основные ресурсы могут использоваться более эффективно, если трафик динамически сбалансирован между несколькими путями.

Переадресация пакетов на конкретные пути: как только источники решили переслать определенное количество пакетов между несколькими путями. Пакет может направляться по определенному пути через туннели. Сегодня используются два распространенных метода туннелирования: IP-in-IP туннелей и Multi Protocol Label Switching (MPLS). В обоих случаях установление туннеля задействует “pushing” дополнительного IP-заголовка на входе в туннель и “popping” IP-заголовка на туннельном выходе. В случае MPLS,

каждый промежуточный маршрутизатор также хранит таблицу пересылки на основе меток, чтобы иметь возможность отправлять пакет по соответствующем внешнем канале, основываясь на его заголовке. Сегодня, маршрутизаторы уже могут разделять трафик поровну между несколькими путями с помощью различных методов [15].

1. 2. Анализ протоколов многопутевой маршрутизации

Уровень приложений

Большинство современных приложений полагаются на транспортный или ниже (т.е. сетевой или канальный) слои для доставки контента с помощью многопутевого выделения ресурсов. В результате, приложения имеют меньше контроля над выбором пути, пересылки данных и т.д., так как эти прерогативы подпадают под область нижних слоев. Таким образом, в мультимедийных приложениях реального времени, многопутевых транспортных протоколов в режиме реального времени – таких, как MPRTP (Multi-Path Real-Time Protocol) [16] – дополнительно требуется удовлетворить SLA (соглашение об уровне сервиса) и QoS требования. Кроме того, развертывание нескольких «похожих» путей для потока приведет к увеличению скорости доставки данных, так как они объединяют потенциал нескольких каналов.

Многопутевая маршрутизация на транспортном уровне

Хотя многопутевое выделение ресурсов могут быть реализованы на различных уровнях, транспортный уровень оказывается лучшим местом для предоставления-соединения из конца в конец заявки по непересекающимся путям. В настоящее время, характеристики одного end-to-end пути оцениваются в основном на транспортном уровне. Следовательно, многопутевое выделение ресурсов на этом слое может использовать эти характеристики для каждого пути для достижения многочисленных преимуществ по сравнению с механизмами одноканального. К тому же, большинство хостов в настоящее время поддерживается более чем одним сетевым интерфейсом, например,

Ethernet, WiFi и сотовой связью. Следовательно, транспортный уровень может использовать их, чтобы установить несколько путей. В работе [17], Hsier и др. предложили pTCP (параллельный TCP) – протокол на основе TCP, в качестве транспортного end-to-end протокола раздеть данных по нескольким путям в беспроводных доменах. Другой протокол на основе TCP для работы с многопутевым распространением является M/TCP [18], который использует “мульти-маршрут” вариант идентичности в заголовке TCP для передачи данных по нескольким путям, и поддерживает “информационные списки” переданных сегментов по каждому маршруту, чтобы преодолеть сложности в обработке подтверждений и повторных передач. MTCP [19] – это расширение TCP – он также разделяет потоки данных по нескольким путям из конца в конец, и развертывает общий механизм обнаружения узких мест для идентификации перегруженных каналов и подавления некоторые из MTCP подпотоков для того, чтобы не использовать неоправданно большую долю пропускной способности по сравнению с другими одноканального TCP потоками. MTCP выбирает ряд кандидатов путей для соединения с использованием лежащих в основе устойчивых наложенных сетей (RON) [20]. Следовательно, он не может быть использован в произвольной сети IP. cTCP (одновременный TCP) другой подход балансировки нагрузки на транспортном уровне, который может быть использован для установления нескольких путей соединения между многосетевыми конечными хостами. Параллельная многопутевая передача (CMT) [21] – это расширение SCTP – использует функцию Многодоменность для одновременной передачи данных по нескольким независимым путям. Было предложено использование единого последовательного пространства (а не отдельно для каждого пути) для ассоциативности всех путей и утверждения выгоды в контроле перегрузки, обнаружения потерь и восстановления данных траффика. Тем не менее, разделение единого последовательного пространства по этим путям оставляет пробелы в нем, и поэтому пакеты могут быть блокированы посредниками на пути, такими как трансляция сетевых адресов (NATs) в интернете [22].

В последнее время Internet Engineering Task Force (IETF) рабочая группа стандартизировала многопутевой протокол для транспортного уровня. Они расширили обычный TCP до многопутевого TCP (MPTCP), что позволяет разделить транспортное соединение на несколько “субпотоков” одновременно и работать по нескольким путям. Эти субпотоки используют либо разные IP-адреса или различные номера портов с одним и тем же IP-адресом (из многосетевых конечных хостов) для обмена пакетами, принадлежащими к одному соединению MPTCP над параллельными (возможно, непересекающихся) путями. MPTCP выглядит как обычное соединение TCP на прикладном уровне. Однако сетевой слой обрабатывает каждый MPTCP субпоток как обычный TCP поток, выхватывая из него дополнительную информацию из опционального поля TCP. MPTCP был разработан, для [23]:

• Повышение пропускной способности – производительность MPTCP должна быть по крайней мере так же высока, как TCP.

• “Не навреди” – MPTCP должен работать во всех сценариях, где в настоящее время работает TCP, и он не должен занимать больше возможностей, чем однопутевой поток.

• Баланс заторов – MPTCP должны смещать траффик с перегруженных путей к менее загруженным из них эффективным разделением подпотоков.

MPTCP начинается как обычный TCP, а затем устанавливает дополнительные потоки, если существует несколько путей между многосетевыми (или мульти-адресными) конечными пользователями. Подпоток, по существу, это – обычный TCP поток MPTCP соединения, который может быть настроен как TCP поток с некоторым дополнительным механизмом сигнализации. Все сигналы MPTCP выполняются в необязательном заголовке поля (40 байт) TCP. Для достижения высокой пропускной способности, MPTCP создаёт отдельный окно насыщения для каждого подпотока, и размер окон изменяется на основании уровня перегрузки, измеряемых на каждом подпотоке Окно перегруженности указывает объем данных, которое отправитель может передавать без по связи. Кроме того, окна загруженности на различных субпотоках соединены на уровне соединения, чтобы гарантировать, что

данные могут быть сдвинуты из перегруженных путей к менее загруженным из них, а суммарная пропускная способность соединения MPTCP не должно быть больше, чем у одного соединение TCP.

В работе [24] для сетей CDNs предложена многопутевая адаптивная схема HTTP с кодированием мультимедийного контента в различных скоростях передачи данных, и разделяет его на сегменты той же продолжительности. Программный агент в пользовательском терминале запрашивает эти сегменты из нескольких серверов HTTP параллельно с использованием HTTP запроса с байтовым диапазоном.

В работе [25] приведен анализ многопутевого протокола MP-TCP, обеспечивающий равномерную загрузку сети передачи данных. В работе [26] предложен многопутевой междоменный протокол маршрутизации на основе протокола BGP.

В работе [27] рассмотрены вопросы многопутевой маршрутизации с использованием протокола TCP, обеспечивающие требуемые параметры качества обслуживания QoS. Рассмотрены вопросы междоменной и внутри доменной маршрутизации, позволяющие создать из конца в конец несколько параллельных путей между конечными узлами сети. В работе [28] представлен протокол транспортного уровня MPTCP, реализованный в рамках ОС IOS от Apple 7, который реализовал Multipath TCP для передачи данных через различные мобильные интерфейсы, такие как Wi-Fi и 4G LTE. В основном, он использует интерфейс Wi-Fi в качестве основного соединения TCP, а также сотовой связи в качестве интерфейса подключения резервного копирования для повышения отказоустойчивости. Параллельные пути из конца в конец могут осуществлять контроль за балансировкой нагрузки и заторами в сети, так что конечные получатели могли бы адаптивно переложить нагрузку с перегруженных путей к менее загруженным из них. Показано, что ресурсоёмкие приложения, будут успешно работать на более высоких пропускных способностях путем агрегирования пути и, соответственно, разделив трафик приложений между этими параллельными путями.

Исследованы многочисленные работы маршрутизации многопутевого распространения на основе архитектуры Path Computation Element (PCE) [29]. В работе [30] представлен способ поиска путей в прямом и обратном направлениях с помощью сигнализации в многодоменных сетях, базирующихся на PCE. В работе [31] представлен способ коммутации на канальном уровне с помощью многопутевого коммутатора (OLIMPS), направленный на повышение эффективности, управляемости, использования больших сетей за счет оптимизации отображения потока данных в сложных многопутевых топологиях.

В работе [32] предложена и проанализирована многопутевая многоадресная схема передачи файла для уменьшения времени передачи для трансфертов один-ко-многим в OpenFlow-сетях.

В работе [33] предложено расширение протокола BGP (путем изменения правил выбора пути) для многопутевой операций.

В последнее время обратная совместимость многопутевого решения BGP, называемая BGP eXtended Multipath (BGP-XM), предложена в работе [34], которая выбирает несколько маршрутов, полученных от различных ASes и рекламирует их с помощью агрегированного атрибута, такого как AS_PATH, в сообщение единого обновления, не нарушая обычной функциональности BGP. В работе [35] сосредоточено внимание на маршрутизации многопутевого пути между доменами, который представляет собой некооперативный многопутевой пиринг фреймворка для координации между двумя поставщиками, учитывая расходы и маршрутизации, и заторов. Они показывают, что политика координации может снизить стоимость маршрутизации приблизительно на 10%, улучшить стабильность маршрута, и избежать заторов. С другой стороны, [99] использует мульти-AS сотрудничество с целью обеспечить балансировку и устойчивость end-to-end соединения.

MPRTP разбивает один поток на несколько подпотоков, которые затем направляются на несколько путей [36]. Он использует протокол управления RTP (RTCP), чтобы контролировать доставку данных и управляющей информации, такой как информация синхронизации, потери, джиттера и т.д. из

конца в конец. Буфер в приемнике компенсирует задержки дифференциального пути и сортирует OUTOF-порядок пакетов. Хотя некоторые специфические механизмы контроля насыщения для приложений реального времени, которые еще предстоит проверить, планировщик использует информацию управления в-пути и время кругового обхода (RTT), чтобы благоразумно смещать траффик между перегруженным, мягко перегруженным, и ненагруженным путями.

Сетевой уровень

В работе [37] предложен алгоритм многопутевой маршрутизации, обеспечивающий равномерную загрузку беспроводной Mesh сети.

В работе [38] фокусируется проблема выбора лучшего пути в беспроводных мультитранзитных сетях. Во-первых, устанавливается непересекающиеся множество путей между исходным узлом и узлом назначения алгоритма обнаружения маршрута; затем, используется теория игр для выбора пути и оптимизация распределения частей на путях; наконец, используется секретная схема распределения, чтобы далее улучшить отказоустойчивость и безопасность. Результаты моделирования показывают, что протокол улучшает безопасность маршрутизации и отказоустойчивость.

В работе [39] проанализированы преимущества использования многопутевой маршрутизации в MANET. Данная технология может увеличить сопротивляемость протокола маршрутизации к атакам с отказом, однако процесс определения маршрута все еще уязвим к атакам. Это исследование приводит к разработке нового алгоритма маршрутизации, который обеспечивает хороший компромисс между поиском максимально непересекающихся путей и перегрузкой сети пакетами. Также представлена новая система безопасности для защиты данного алгоритма маршрутизации. Основной задачей является обеспечить высокий уровень безопасности при одновременном снижении нагрузки на сеть, если атаки нет. Ранние эксперименты показали, что так как перегрузки при ответах и защита от ошибок не являются оптимальными, представленная схема двойного шифрования эффективно снижает уровень

нагрузки на сеть при рассылке запросов на маршрутизацию. Криптосистема RSA-TS довольно ресурсоёмкая и не является оптимальной для MANET.

В работе [40] предложен эвристический алгоритм многопутевой маршрутизации, обеспечивающий балансировку нагрузки на уровне приложений, что позволяет обеспечить заданные параметры QoS, такие как задержка передачи мультимедийных данных.

В работе [41] предложен многопутевой маршрутный протокол с вероятностными непересекающимися узлами, способный к восстановлению(SNMR), который формирует вероятностные непересекающиеся маршруты вместе с альтернативными путями для промежуточных узлов в основном пути. С этой целью протокол использует фильтр Блума для хранения информации по основному пути с минимальными затратами. Кроме того, альтернативные пути можно найти для всех промежуточных узлов в основного пути, одновременно. Как следствие, SNMR обеспечивает более высокую устойчивость к мобильности и имеет лучшую способность восстановления пакетов.

В работе [42] предложен алгоритм формирования множества непересекающихся путей при доменной организации структуры мобильной компьютерной сети.

В работе [43] предложен протокол сетевого уровня Locator/Identifier Separation Protocol (LISP), который делит функциональные возможности IP на две части для локализации и отождествления устройства, путем присвоения двух IP-адресов в основном по причине масштабируемости. Локализация IP-адреса, как правило, относится к пограничным маршрутизаторам, которые называются Routing LOCators (RLOCs), и отождествление IP-адреса конечных точек называется Endpoint Identifiers (EIDs). LISP граничные маршрутизаторы оповещают о своих RLOCs таким образом, что EID может достигать их, и создавать туннели для пересылки пакетов конечных станций между RLOCs маршрутизаторами, предваряя IP-адрес назначения RLOC. LISP граничные маршрутизаторы могут разделить поток по нескольким путям с использованием адресов различного назначения RLOCs.

В работе [44] предложен протокол многопутевой маршрутизации, основанный на поведении муравьев. Формирование множества маршрутов осуществляется с помощью мультиагентной системы. С помощью данного алгоритма осуществляется формирование локального запроса маршрута каждый раз, когда узел планирует отправить пакет данных. также пытается решить несколько проблем, таких как: процент доставки пакетов, сквозная задержка, время жизни сети и потребление энергии. С этой целью предлагается гибридный метод, который влечет за собой превентивные и реактивные процессы. Маршруты устанавливаются и периодически сохраняются с постоянным количеством мобильных агентов. Агент периодически создается каждым узлом, таким образом количеством агентов в сети можно все время управлять. Однако, когда запланировано, что соединение должно быть установлено узлом с другим узлом при отсутствии маршрута в его таблице маршрутизации, последний выполнит запрос маршрута, устанавливая локальную переменную, доступную для агентов, проходящих через него. Основным недостатком данного алгоритма является относительно низкая его сходимость, свойственная всем самоорганизующимся алгоритмам.

1.3. Способы формирования множества путей

По типу математического подхода способы формирования множества путей делятся на комбинаторные и потоковые [45].

В основу комбинаторных алгоритмов положено математическое описание компьютерных сетей в виде ориентированного или неориентированного графа с последующим использованием комбинаторных алгоритмов поиска множества кратчайших путей между заданными парами узлов (вершин) сети. К таким методам можно отнести алгоритмы, описанные в работах [46,47,48,49,50].

К комбинаторным алгоритмам относятся алгоритмы поиска в глубину (англ. Depth-first search, DFS) [51], которые базируются на рекурсивном об-

ходе вершин графа, выбирая на каждом шаге вершину с минимальным значением заданной метрики.

К потоковым алгоритмам относятся алгоритмы [52,53,54]. В рамках потоковых моделей задача маршрутизации, зачастую формулируется в виде задачи математического программирования [55] с использованием технологии распределения потока на графах.

Временная сложность формирования одного виртуального канала с помощью известных комбинаторных алгоритмов, например, алгоритма Дейкстры равна $O(N^2)$, где N – число узлов исходной физической сети в рамках которой формируется VPN. Временная сложность формирования k виртуальных каналов равна $O(kN^2)$, для полносвязной топологии эта величина равна $O(M^2N^2)$.

В общем случае одному виртуальному каналу может соответствовать несколько физических, как правило, непересекающихся каналов. Это позволяет повысить эффективность процедуры конструирования трафика (TE), обеспечить заданные параметры QoS при изменении параметров сети, в частности при изменении метрики каналов передачи информации, путем замены одного физического канала другим в рамках данного виртуального канала [56,57]. В общем случае временная сложность формирования одного виртуального канала, состоящего из q физических каналов равна $O(qN^2)$.

При формирования виртуального канала, состоящего из множества непересекающихся путей вершины, входящие в предыдущие пути не учитываются. В этом случае временная сложность определения по формуле:

$$O\left(N^2 + \sum_{i=2}^{k} \left(N - \sum_{j=1}^{k-1} N_{Lj} \right)^2 \right),$$

где : N – число узлов в сети;

k – количество непересекающихся путей ;

N_{Lj} – количество вершин пути Lj.

В связи с этим актуальным является разработка алгоритмов формирования каналов, характеризующихся минимальной временной сложностью.

В работе [58] рассмотрен способ формирования множества непересекающихся путей и предложен критерий формирования оптимального набора параллельных путей из множества непересекающихся путей между абонентами мобильной компьютерной сети. На начальном этапе пути разделяются на группы, затем из каждой группы выбираются пути с минимальным ожидаемым временем передачи запрошенных данных при условии одновременной передачи.

1.4. Способы многопутевой маршрутизации в распределенных центрах обработки данных

В работе [59] отмечается, что способы однопутевой маршрутизации не эффективны в распределенных центрах данных. Протоколы маршрутизации в распределенных центрах данных должны быть в состоянии активно (или реактивно) разгрузить узкие места каналов путём перенаправления некоторого траффика на каналы с недостаточным уровнем использования. Применение многопутевых протоколов маршрутизации позволяет в полной мере учитывать структуру распределенных центров данных для улучшения балансировки нагрузки и уменьшения потери пакетов.

В настоящее время имеется ряд алгоритмов многопутевой маршрутизации для центров обработки данных. Большинство из них ориентированы на конкретные сетевые топологии. В работе [60] предложен алгоритм распределенной многопутевой маршрутизации для центров обработки данных для сетевых топологий FatTree, основанный на методе линейного программирования. В работе [61] на основе метода Лагранжа предложен способ распределенной многопутевой маршрутизации. В работе [62] предложен распределенный алгоритм многопутевой маршрутизации для центра обработки данных сети на основе стохастического моделирования, который уравновешивает передачу между несколькими каналами путем минимизации вероятности. Данная задача является NP-сложной задачей для топологий FatTree, VL2,

BCube, и DCell [63], которые используются для организации распределенных центров обработки данных. В большинстве случаев трафик центра обработки данных моделируется с помощью распределения с тяжелым хвостом (heavy-tailed distributions (HTD)) к которым относится распределение Парето.

Многопутевые связи было настоятельно рекомендуемы для центров обработки данных, сетей для лучшей перспективы использования сетевых ресурсов. В работе [64] предложили SDN на основе балансировки нагрузки для fat-tree центров обработки данных с многопутевой поддержкой.

Многопутевой TCP также применяется в центрах обработки данных [65]. С этой целью, MPTCP сравнивается с PacketScatter (распространяет пакеты потока по нескольким путям), обычный TCP и два варианта TCP (несвязанный TCP и равновзвешанный TCP) применяются на основе моделирования топологии FatTree [66]. Хотя PacketScatter достигает наивысшей пропускной способности и лучшей балансировки нагрузки, порядок доставки остается проблемой, особенно когда длины самых коротких и длинных путей сильно отличаются. Всё же MPTCP превосходит стандартный протокол TCP.

В работе [67] предложен масштабируемый распределенный алгоритм планирования потоков данных в сетевых центрах данных. Данный алгоритм основывается на методах линейного программирования.

В работе [68] представлен модифицированный протокол PEFT в который включен модуль измерения интернет-трафика для вычисления оптимального набора весов каналов, а также для мониторинга использования линий связи отдельных звеньев. Модифицирована PEFT обеспечивает лучшую балансировку нагрузки и использование линии связи по сравнению с ECMP на основе дерева топологий (каноническое дерево, толстое дерево, сцепленное дерево).

В работе [69] предложен способ балансировки нагрузки с помощью алгоритма многопутевой маршрутизации, который может лучше распределить нагрузку трафика и использовать доступную полосу пропускания в сети с топологией FatTree. Данная задача сформулирована в виде системы линейного программирования. В качестве практического решения, рассмотрена ба-

лансировка нагрузки с помощью алгоритма многопутевой маршрутизации для SDN на базе центров обработки данных сетей. Алгоритм использует центральный контроллер для сбора информации о состоянии сети, и делает решения оптимизированной балансировки нагрузки при приеме новых потоков в сеть.

1.5. Формирование виртуальных грид систем.

Технология облачных вычислений быстро развивается, и становится дефакто интернет-стандартом вычислений, [70] хранения и инфраструктурой размещения, как в промышленности, так и в научных кругах. Огромные перспективы масштабируемости, предлагаемые облачной инфраструктурой, могут быть использованы не только для сервисов и хостинга приложений, но и как вычислительный ресурс грида по требованию. Облачная инфраструктура может быть использована для обеспечения грид-ресурсов по требованию с поддержкой виртуальных машин (VM) и известна как монитор виртуальных машин. Одна из целей Грида – вычисления для огромного количества приложений с поддержкой использования виртуальных ресурсов [71]. В работе [72] рассмотрены вопросы построения виртуальной Grid –системы, ориентированной на многопутевую маршрутизацию.

В работе [73] рассмотрены основные концепции и способы организации облачных вычислений на основе VPN в распределенных центрах данных. В табл. 1.2 приведена классификация центров обработки данных с учетом их уровня виртуализации [74]:

Таблица 1.2

Система	Уровень масштабируемости	Гарантия производительности	QoS	Отказоустойчивость
VL2 [75]	Высокий	нет	нет	да
SecondNet [76]	Высокий	да	да	да
PortLand [77]	Высокий	нет	нет	да
Gatekeeper [78]	Высокий	да	да	да

CloudNaaS [79]	Низкий	нет	да	да
Oktopus [80]	Высокий	да	да	да
NetLord [81]	Высокий	нет	нет	нет
Seawall [82]	Высокий	нет	нет	да

В работе [83] на основе анализа известных многопутевых протоколов предложен модифицированный протокол многопутевой маршрутизации, позволяющий повысить эффективность конструирования трафика в компьютерных сетях большой размерности.

ПОСТАНОВКА ЗАДАЧИ ИССЛЕДОВАНИЯ

1. Существенным недостатком существующих комбинаторных способов многопутевой маршрутизации является их значительная временная сложность, которая напрямую зависит от числа формируемых маршрутов и большой объем памяти для хранения таблиц маршрутизации, в связи с этим актуальным является задача определения на начальном этапе маршрутизации максимально возможного числа допустимых маршрутов.

2. По сравнению с комбинаторными алгоритмами маршрутизации потоковые алгоритмы характеризуются меньшей временной сложностью. Это определяет целесообразность разработки и использования потоковых алгоритмов многопутевой маршрутизации.

ГЛАВА 2

ФОРМИРОВАНИЕ МНОГОПУТЕВЫХ ВИРТУАЛЬНЫХ КАНАЛОВ

2.1. Определение максимального числа непересекающихся путей между двумя начальной и конечной вершинами виртуального канала

Большинство известных методов формирования множества непересекающихся путей основаны на методах направленного перебора возможных путей. При этом предварительное определение максимально возможного числа непересекающихся путей позволяет сократить количество операций перебора и, соответственно, уменьшить временную сложность формирования путей.

Количество непересекающихся путей зависит от топологии графа сети, в частности от характера области сочленения графа, от числа вершин и их степени. В общем случае максимальное количество непересекающихся путей между двумя вершинами $в_i$ и $в_j$ определяется минимальным значением степени вершин $в_i$, $в_j$ и минимальным множеством сочленения графа $G=(B,E)$. В данном случае под минимальным множеством сочленения $B_S = B\setminus(B_1 \cup B_2)$ графа будем понимать множеством вершин, удаление которых делит граф $G=(B,E)$ на два подграфа $G_1=(B_1,E_1)$ и $G_2=(B_2,E_2)$ при условии, $в_i \in B_1$, а $в_j \in B_2$.

Рассмотрим условия существования максимального множества непересекающихся путей между начальной и конечной вершинами виртуального канала.

Пути P_i и P_j не пересекаются при условии $B_i \cap B_j = \varnothing$, где: B_i – множество вершин пути P_i, а B_j – множество вершин пути P_j.

При формировании множества непересекающихся путей между начальной вершиной $в_0$ и конечной вершиной $в_n$ граф G (B,E) компьютерной сети представляется в виде двух подграфов $G_1(B_1,E_1)$ и $G_2(B_2,E_2)$, при этом $в_0 \in B_1$, а $в_n \in B_2$. Множество $B_1=B^i_1 \cup B^o_1$, где: $B^i_1 = \{в_i |\ i=1,2,\ldots,n\}$ – внутренние вершины подграфа $G_1(B_1,E_1)$.

Для множества внутренних вершин $B^i_1 \subset B_1$ выполняется условие $E^i=\{e^i_{k,j}|$ $в_k \in B_1$, $в_j \in B_1\}$. Соответственно ребро $e^i_{k,j}$ является внутренним ребром.

Вершины множества $B^o_1=\{в^o_i|\ i=1,2,\ldots,n\}$, смежные с вершинами множества B_s, будем называть граничными вершинами подграфа $G_1(B_1,E_1)$. Удаление всех граничных вершин приводит к тому, что граф становится несвязным. Соответственно ребро $e^o_{k,j}$ будем называть граничным ребром. Для множества граничных вершин $B^o_1 \subset B_1$ выполняется условие $E^o = \{e^o_{k,j}|\ в_k \in B_1$, $в_j \in B_2\}$.

Будем различать внутреннюю S^i_k и внешнюю S^o_k степени вершины. Количество внутренних ребер вершины $в_k$ определяет ее внутреннюю степень S^i_k. В свою очередь ребро $e^o_{k,j} = \{в_k \in B_1, в_j \notin B_1\}$ является внешним для подграфа $G_1(B_1,E_1)$, при этом вершина $в_k \in B_c$. Количество внешних ребер вершины $в_k$ определяет ее внешнюю степень S^o_k. Для вершин $B^i_1 = \{в_i|\ i=1,2,\ldots,n\}$ внешняя степень $S^o_k=0$.

Лемма 1. Максимальное количество непересекающихся путей $N_{\max} = \min\{S_i, S_j, \min|B_s|_n\}$, где: S_i – степень вершины $в_i$; $\min|B_s|_n$ – мощность минимального множества соединения графов.

При S_i и $S_j \geq \min|B_s|_n$ максимальное количества непересекающихся путей определяется величиной $\min|B_s|_n$. В этом случае каждая из вершин $в_i \in \min B_s$ обязательно находится на одном из путей между вершинами $в_0$ и $в_n$.

2.2. Условие формирования множества непересекающихся путей

Известные комбинаторные и графические методы формирования путей на основе различных метрик не гарантируют формирования множества непересекающихся путей. Например, для графа, представленного на рис 2.1. при формирования путей на основе метрики, оптимизирующей минимальное количество переходов формируется только один минимальный непересекающийся путь $P_1 = \{в_0,в_2,в_8,в_{13},в_{21},в_{22}\}$. Все остальные пути будут пересекаться с данным путем.

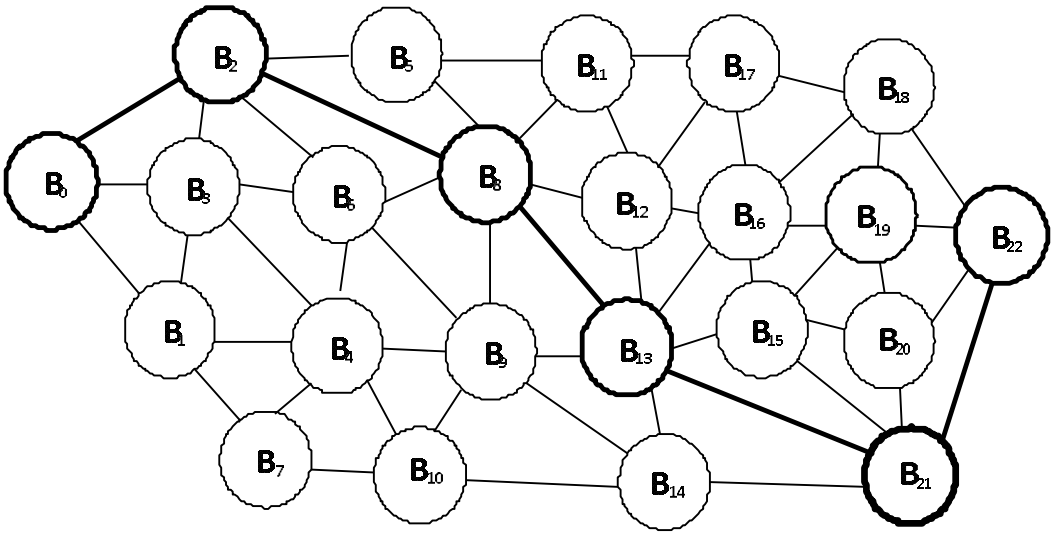


Рис 2.1 Формирование минимального пути

В общем случае для графа, представленного на рис. 2.1, существует три непересекающихся пути $P_1=\{в_0,в_2,в_5,в_{11},в_{17},в_{18},в_{22}\}$, $P_2=\{в_0,в_3,в_6,в_8,в_{12},в_{16},в_{19},в_{22}\}$ и $P_3=\{в_0,в_1,в_7,в_{10},в_{14},в_{21},в_{22}\}$.

Формирование очередного непересекающегося пути эквивалентно удалению соответствующих вершин и связанных с ними ребер и разбиению графа на два подграфа. При этом удаление вершин с минимальной степенью увеличивает связность сформированных подграфов и обеспечивает возможность формирования новых непересекающихся путей. На основании этого можно сформулировать условие формирования условие формирования максимального количества непересекающихся путей следующим образом.

Лемма 2. Необходимым условием формирования максимального количества непересекающихся путей между вершинами $в_0$ и $в_n$ является:

$$\min \sum_{i=1}^{Nmax}\left(\sum_{j=1}^{L_i} S_j^0\right), \qquad (2.1)$$

где: L_i – длина пути P_i между вершинами $в_0$ и $в_n$;

N_{max} – максимально возможное число непересекающихся путей;

S^o_j – внешняя степень вершины $в_j \in P_i$.

Лемма 2 определяет следующие пункты правила формирования максимального множества непересекающихся путей:

1. Выбор пути P_i с минимальным значением S^o_k для текущей вершины $в_k$.

2. При равных значениях внешней степени выбирается путь с минимальным весом $W_i = \sum_{j=1}^{L_i} S_j^0$.
3. В качестве очередной вершины пути выбирается смежная с вершиной $в_k$ вершина $в_m \notin B_l$ с минимальной внешней степенью S^o_m.

2.3. Потоковый алгоритм формирование множества непересекающихся путей между двумя узлами

Временная сложность формирования многоканального виртуального пути, состоящего из q физических каналов с помощью комбинаторных алгоритмов, например, алгоритма Дейкстры равна $O(qN^2)$. Уменьшить временную сложность можно за счет исключения операции перебора вариантов на основе метода «ветвей и границ», однако данный алгоритм не позволяет учитывать метрики пути при организации многопутевой маршрутизации. В связи с этим, в работе [84] был предложен модифицированный волновой алгоритм многопутевой маршрутизации (рис. 2.2). Особенностью данного модифицированного алгоритма является возможность сформировать все возможные непересекающиеся пути с учетом их метрик.

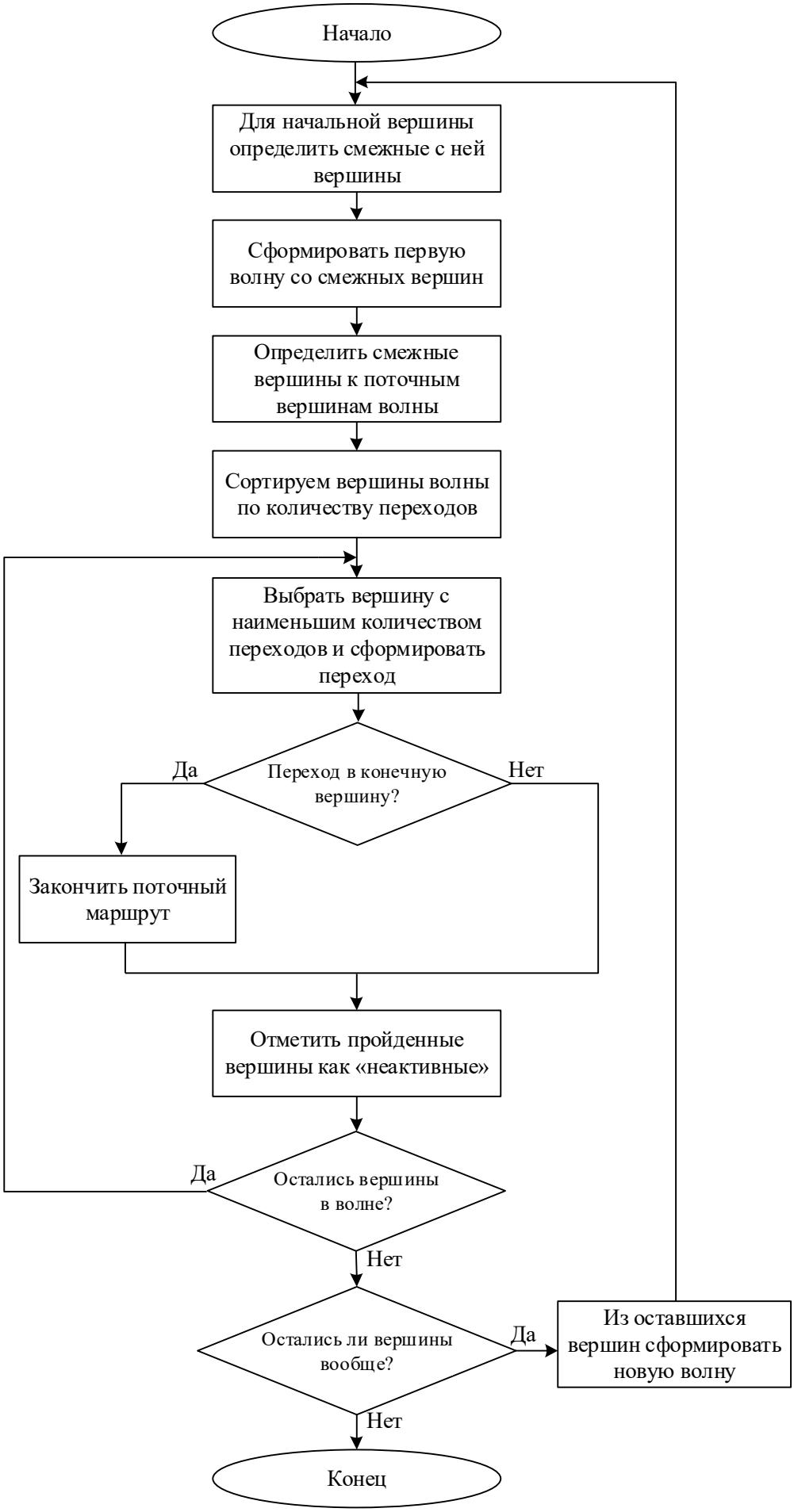


Рис. 2.2 блок-схема модифицированного волнового алгоритма

Рассмотрим пример модифицированного волнового алгоритма формирования множества путей между начальной вершиной B_0 и конечной вершиной $в_{17}$ (рис.2.3).

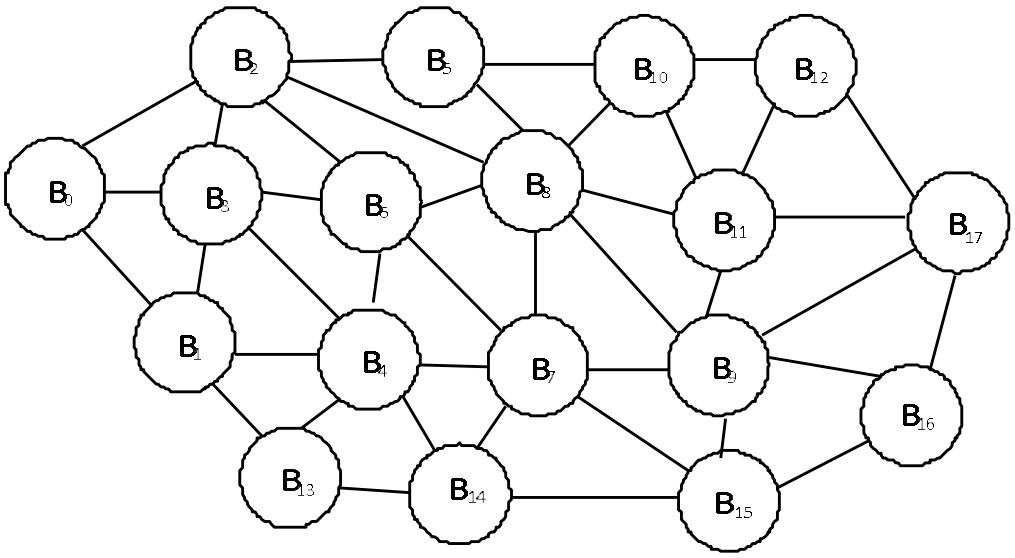


Рис. 2.3 Граф сети

На начальном этапе строится дерево из начальной вершины и смежных с ней вершин. Например, для графа, представленного на рис. 2.3, для начальной вершины B_0 смежными являются вершины $в_1, в_2, в_3$. Таким образом на первом шаге формируются пути $P_1=\{в_0, в_1\}$, $P_2=\{в_0, в_3\}$, $P_3=\{в_0, в_2\}$ (рис. 2.4) с начальным уровнем в вершине $в_0$ и множеством вершин первого уровня $L_1=\{в_1, в_2, в_3\}$.

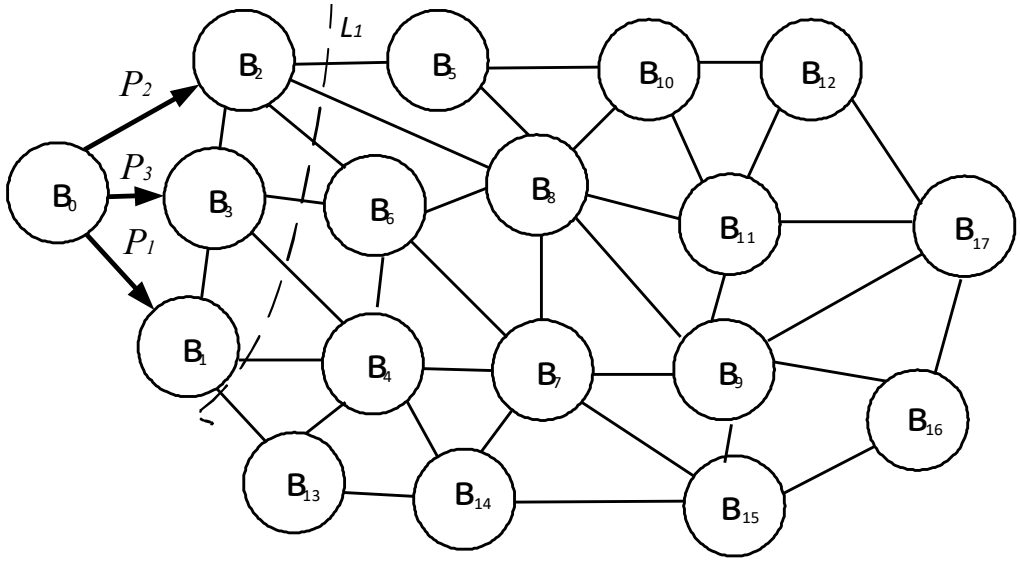


Рис.2.4 Начальный уровень формирования пути

На следующем шаге пути продлеваются до второго уровня $L_2=\{в_4,в_5,в_6,в_8,в_{13}\}$ (рис 2.5). При этом соблюдается правило выбора очередной вершины для продления пути (Лемма 2). В данном случае вершины $в_1$ и $в_3$, имеют равное значение внешней степени, поэтому берется первая вершина $в_1$ из множества $L_1=\{в_1,в_2,в_3\}$. Путь $P_1=\{в_0,в_1\}$ разветвляется на два пути $P_{1,1}=\{в_0,в_1,в_{13}\}$ и $P_{1,2}=\{в_0,в_1,в_4\}$.

На следующем шаге продлевается путь $P_3=\{в_0,в_3\}$ от вершины $в_3$ до вершины $в_6$, так как вершина имеет единственный путь до множества $L_2=\{в_4,в_5,в_6,в_8,в_{13}\}$ вершин второго уровня и становится равным $P_3=\{в_0,в_3,в_6\}$.

Путь $P_2=\{в_0,в_2\}$ разветвляется на два пути $P_{2,1}=\{в_0,в_2,в_5\}$ и $P_{2,2}=\{в_0,в_2,в_8\}$.

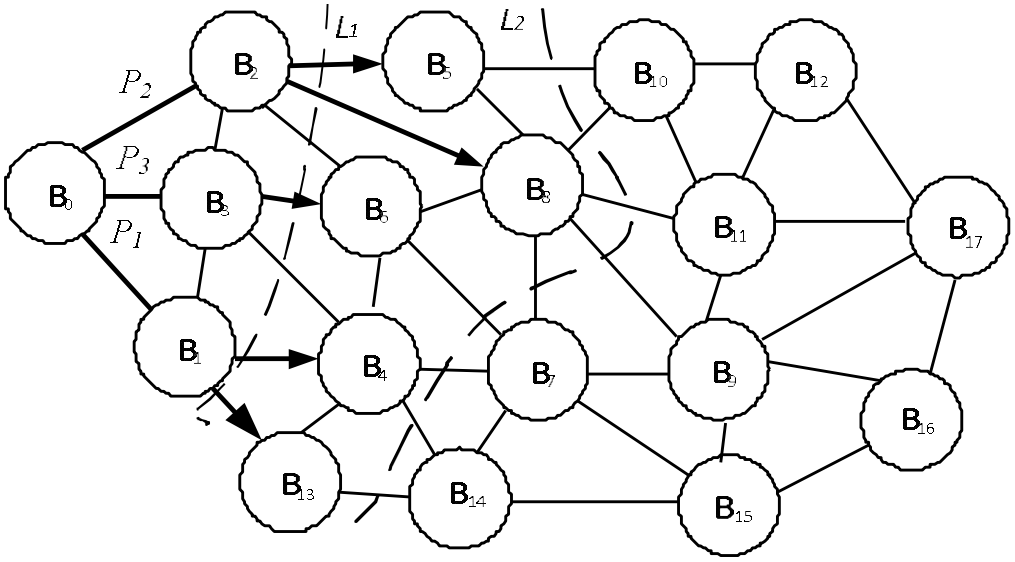


Рис 2.5 Второй уровень формирования дерева путей

На следующем шаге пути продлеваются до третьего уровня $L_3=\{в_7,в_{10},в_{11},в_{14}\}$ (рис 2.6). В данном случае вершины $в_5$ и $в_{13}$, имеют равное значение внешней степени, поэтому берется первая вершина $в_5 \in P_{2,1}$ из множества $L_2=\{в_4,в_5,в_6,в_8,в_{13}\}$. Путь $P_{2,1}=\{в_0,в_2,в_5\}$ продлевается до свободной вершины $в_{10}$ третьего уровня вершин и становится равным $P_{2,1}=\{в_0,в_2,в_5,в_{10}\}$.

Затем берется вершина $в_{13} \in P_{1,1}$ и путь $P_{1,1}=\{в_0,в_1,в_{13}\}$ продлевается до вершины $в_{14}$ и становится равным $P_{1,1}=\{в_0,в_1,в_{13},в_{14}\}$.

Затем выбирается вершина $в_8 \in P_{2,2}$ и путь $P_{2,2}=\{в_0,в_2,в_8\}$ продлевается до свободной вершины $в_{11}$ третьего уровня вершин и становится равным $P_{2,2}=\{в_0,в_2,в_8,в_{11}\}$.

Последней выбирается вершина $в_6 \in P_3$ и путь $P_3=\{в_0,в_3,в_6\}$ продлевается до свободной вершины $в_7$ третьего уровня вершин $L_3=\{в_7,в_{10},в_{11},в_{14}\}$ и становится равным $P_3=\{в_0,в_3,в_6,в_7\}$.

Таким образом, формируются три непересекающихся пути P_1, P_2 и P_3, при этом путь P_2 является разветвляющимся на два подпути $P_{2,1}=\{в_0,в_2,в_5,в_{10}\}$ и $P_{2,2}=\{в_0,в_2,в_8,в_{11}\}$.

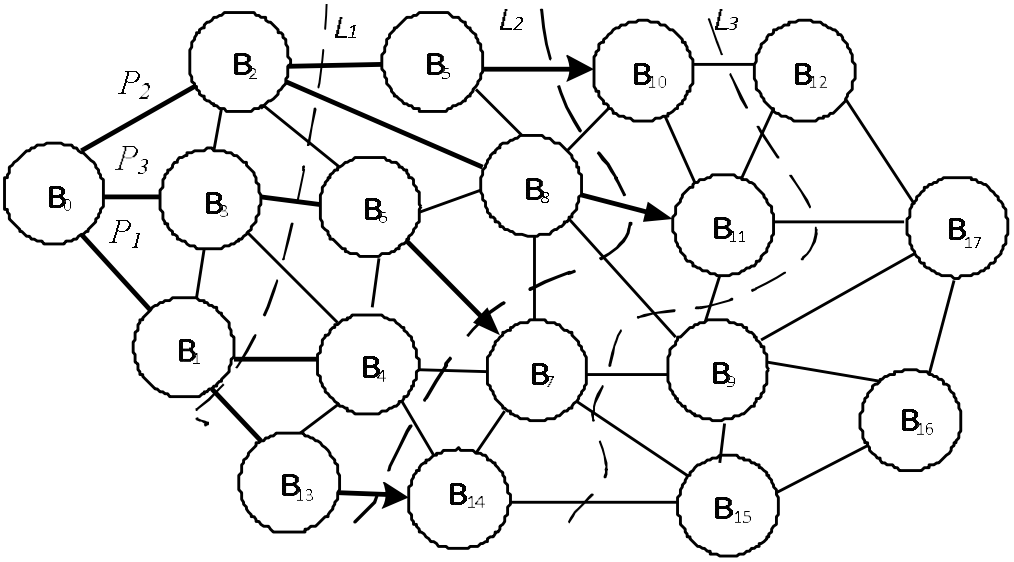


Рис 2.6 Третий уровень формирования путей

На следующем шаге путь продлеваются до четвертого уровня $L_4=\{в_9,в_{12},в_{15}\}$ (рис 2.7). В данном случае вершина $в_{10} \in P_{2,1}$ и $в_{14} \in P_{1,1}$, имеют равное значение внешней степени равное единице, поэтому путь $P_{2,1}=\{в_0,в_2,в_5,в_{10}\}$ продлевается до вершины $в_{12}$ и становится равным $P_{2,1}=\{в_0,в_2,в_5,в_{10},в_{12}\}$. Затем путь $P_{1,1}=\{в_0,в_1,в_{13},в_{14}\}$ продлевается до вершины $в_{15}$ и становится равным $P_{1,1}=\{в_0,в_1,в_{13},в_{14},в_{15}\}$.

Соответственно, путь $P_3=\{в_0,в_3,в_6,в_7\}$ продлевается до вершины $в_9$ и становится равным $P_3=\{в_0,в_3,в_6,в_7,в_9\}$. Путь $P_{2,2}=\{в_0,в_2,в_8,в_{11}\}$ продлевается до конечной вершины $в_{17}$ и формируется путь $P_2=\{в_0,в_2,в_8,в_{11},в_{17}\}$ между начальной $в_0$ и конечной $в_{17}$ вершинами пути.

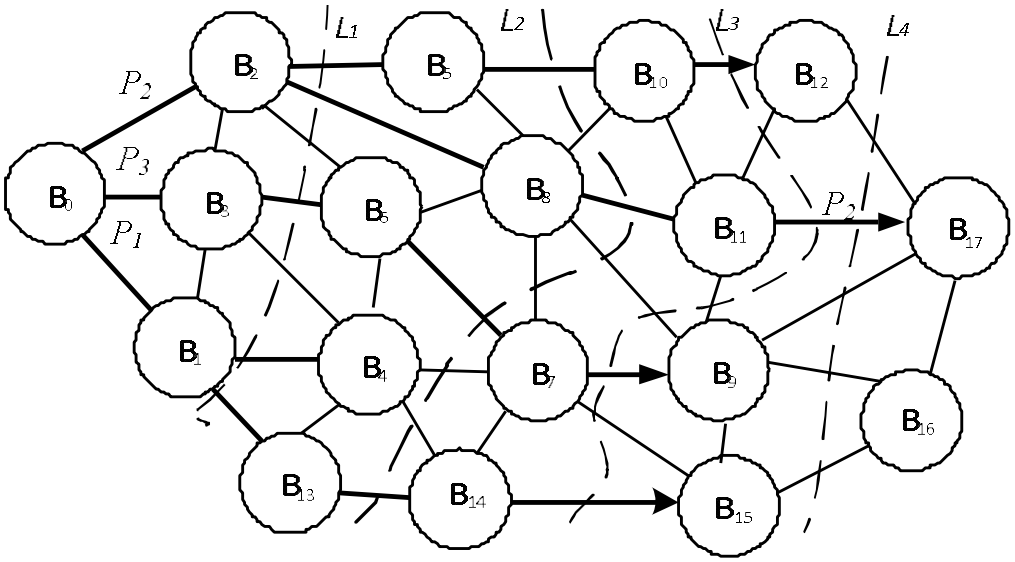


Рис 2.7 Четвертый уровень формирования путей

Затем последовательно формируются пути P_3={в0,в3,в6,в7,в9,в17} и $P_{1,1}$={в0,в1,в13,в14,в15,в16,в17} (рис.2.8). На этом формирование непересекающихся путей заканчивается.

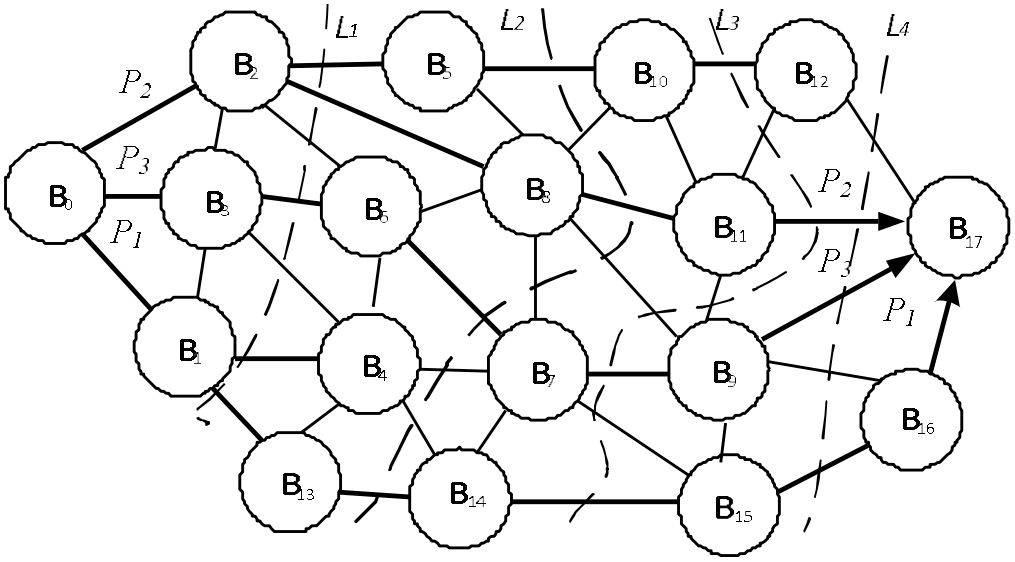


Рис. 2.8 Заключительный этап формирования множества путей

Наличие альтернативных путей позволяет формировать множество непересекающихся путей с учетом их метрик. В частности, вместо пути P_1={в0,в1,в13,в14,в15,в16,в17} на уровне L_2 может быть сформирован путь $P_{1,1}$={в0,в1,в4,в14,в15,в16,в17}, а на уровне L_3 вместо пути $P_{2,2}$={в0,в2,в8,в11,в17} может быть продлен до конечной вершины путь $P_{2,1}$={в0,в2,в5,в10,в12,в17}. Путь P_3={в0,в3,в6,в7,в9} остается без изменений (рис. 2.9). Выбор альтернативного пути определяется его метрикой и принадлежностью к смежному пути, то есть имеющему с ним общие вершины.

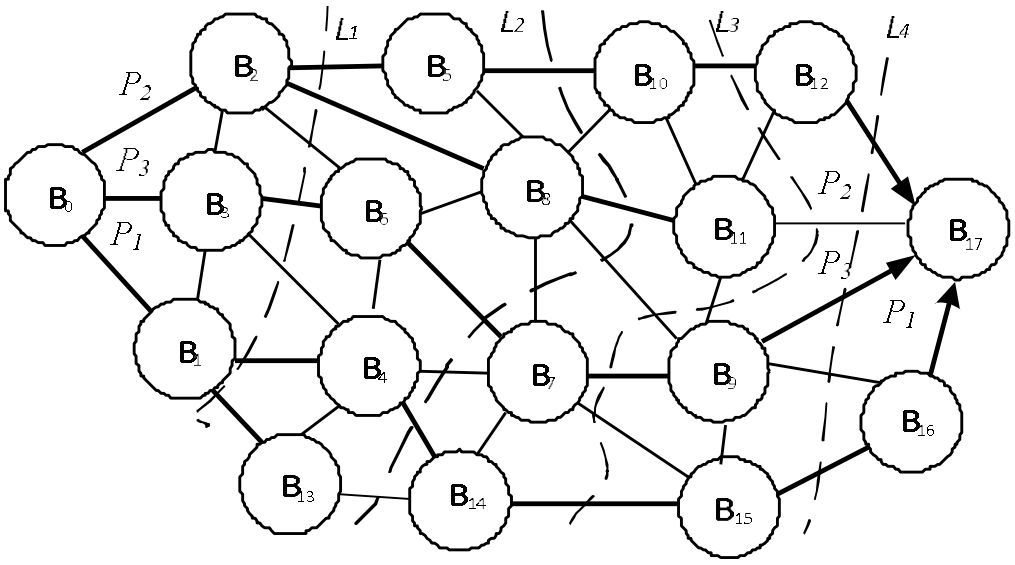


Рис. 2.9 Формирование путей с учетом метрики

В приложении 1 приведена последовательность формирования путей, представленных на рис.2.9, с помощью, разработанной в рамках диссертационной работы программы моделирования модифицированного волнового алгоритма (рис. 2.2).

2.4. Формирование множества непересекающихся путей с учетом минимального множества сочленения графа

В работе [85] предложен алгоритм определения минимального множества сочленения, который основан на процедуре формирования множеств сочленения двух подграфов $G_1=(B_1,E_1)$ и $G_2=(B_2,E_2)$ графа $G=(B,E)$ при котором количество вершин N_1 подграфа $G_1=(B_1,E_1)$ изменяется от 1 до (N-1). В результате этого последовательно формируется несколько различных множеств B_S, среди которых выбирается B_{Smin} с минимальной мощностью h_{Smin}.

При формировании множеств сочленения в подграфе $G_1=(B_1,E_1)$ будем различать внутренние и граничные вершины.

Вершины множества $B^i{}_1=\{в_i|\ i=1,2,\ldots,n\}$ подграфа $G_1(B_1,E_1)$ не смежные с вершинами разделяющего множества B_s будем относить к внутренним вершинам. Для множества внутренних вершин $B^i{}_1 \subset B_1$ выполняется условие

$E^i=\{в^i_{k,j}|\ в_k\in B_1,\ в_j\in B_1\}$. Соответственно ребро $e^i_{k,j}$ является внутренним ребром.

Вершины множества $B_b=\{в^b_i|\ i=1,2,\ldots,n\}$, смежные с вершинами множества B_s, будем называть граничными вершинами подграфа $G_1(B_1,E_1)$. Соответственно ребро $e^b_{k,j}$ будем называть граничным ребром. Для множества граничных вершин $B_b\subset B_1$ выполняется условие $E^b=\{\ e^b_{k,j}|\ в_k\in B_1,\ в_j\in B_s\}$.

Будем различать внутреннюю S^i_i и внешнюю S^b_i степени вершины $в_i$.

Количество внутренних ребер вершины $в_k$ определяет ее внутреннюю степень S^i_k. В свою очередь ребро $e^b_{k,j}=\{в_k\in B_1,\ в_j\notin B_1\}$ является внешним для подграфа $G_1(B_1,E_1)$, при этом вершина $в_k\in B_c$. Количество внешних ребер вершины $в_k$ определяет ее внешнюю степень S^b_k.

Процесс определения минимального множества сочленения B_{Smin} заключается в последовательном формировании множества вершин B_S и определении среди них B_{Smin}, а именно:

1. Из вершин, смежных с начальной вершиной $в_i$, формируется множество вершин B_S, которое в данном случае является B_{Smin}.
2. Множество смежных вершин включается в подграф $G_1=(B_1,E_1)$.
3. Формируется новое множество вершин $B_1=B_1\cup B_S$ подграфа $G_1=(B_1,E_1)$.
4. Формируем новое множество граничных вершин B^o_1.
5. На основе вершин $в_i\notin B_1$ смежных с вершинами множества B^o_1 формируется множество вершин B_S.
6. Вычисляется мощность h_S множества вершин B_S.
7. Сравниваем мощность h_S множества B_S с мощностью h_{Smin} множества B_{Smin}. Если $h_{Smin}>h_S$ то множество сочленения B_S становится B_{Smin}.
8. Формируется граф $G_2=(B_2,E_2)$ с новым множеством вершин $B_2=B_2\backslash B_S$.
9. Если $B_2\neq\{в_i\}$ то переход к пункту 2.
10. Конец.

Рассмотрим процесс определения минимального множества сочленения при формировании пути между вершинами $в_0$ и $в_{17}$ (рис.2.1). На первом этапе

формируется $B_{Smin}=B_S=\{в_1,в_2,в_3\}$, затем формируется $B_S=\{в_8,в_4,в_{13}\}$. На следующем этапе формируется $B_S=\{в_8,в_7\}$, которое является минимальным $B_{Smin}=B_S=\{в_8,в_7\}$. Следующее значение $B_S=\{в_{10},в_{11},в_9,в_{15}\}>B_{Smin}$. Таким образом, остается $B_{Smin}=\{в_8,в_7\}$.

Следует отметить, что между начальной вершиной $в_i$ и вершинами множества сочленения B_S может существовать несколько непересекающихся путей, количество которых больше или равно мощности множества B_{Smin}. Между вершинами множества сочленения B_S и конечной вершиной $в_j$ также может существовать несколько непересекающихся путей.

Уменьшить временную сложность можно за счет формирования путей от исходных вершин до вершин минимального множества сочленения графа. В этом случае временная сложность формирования непересекающихся путей будет определятся временной сложностью формирования путей в каждом их подграфов. Например, при формировании одного пути с помощью алгоритма Дейкстры равна $O(N^2)$, где N – количество вершин графа, а при разделении графа на два подграфа с числом вершин K и $(N-K)$ временная сложность формирования одного пути будет равна $O(K^2+(N-K)^2)$ или $O(2K^2+N^2-2KN)$. Например, при разделении графа с числом вершин сети N=100 на две равных подсети временная сложность соответственно будет равна 2500, а без разбиения на подсети временная сложность будет равна 10000. То есть в четыре раза меньше.

Наличие минимального множества сочленения графа характерно для топологии компьютерных сетей большой размерности. В этом случае компьютерная сеть разделяется на домены маршрутизации и представляется в виде подграфов.

Рассмотрим формирование множества непересекающихся путей на примере графа, представленного на рис. 2.10. В данном случае степень начальной вершины $в_0$ равна 3, степень конечной вершины $в_{17}$ равна 4, а степень минимального множества сочленения $B_S=\{в_7,в_8\}$ равна 2. Таким образом, максимальное число непересекающихся путей между вершинами $в_0$ и $в_{17}$ будет равно 2.

Выбор того или иного набора путей зависит от требований, предъявляемых к качеству обслуживания (QoS) передаваемой информации, и эффективности функционирования сети передачи информации.

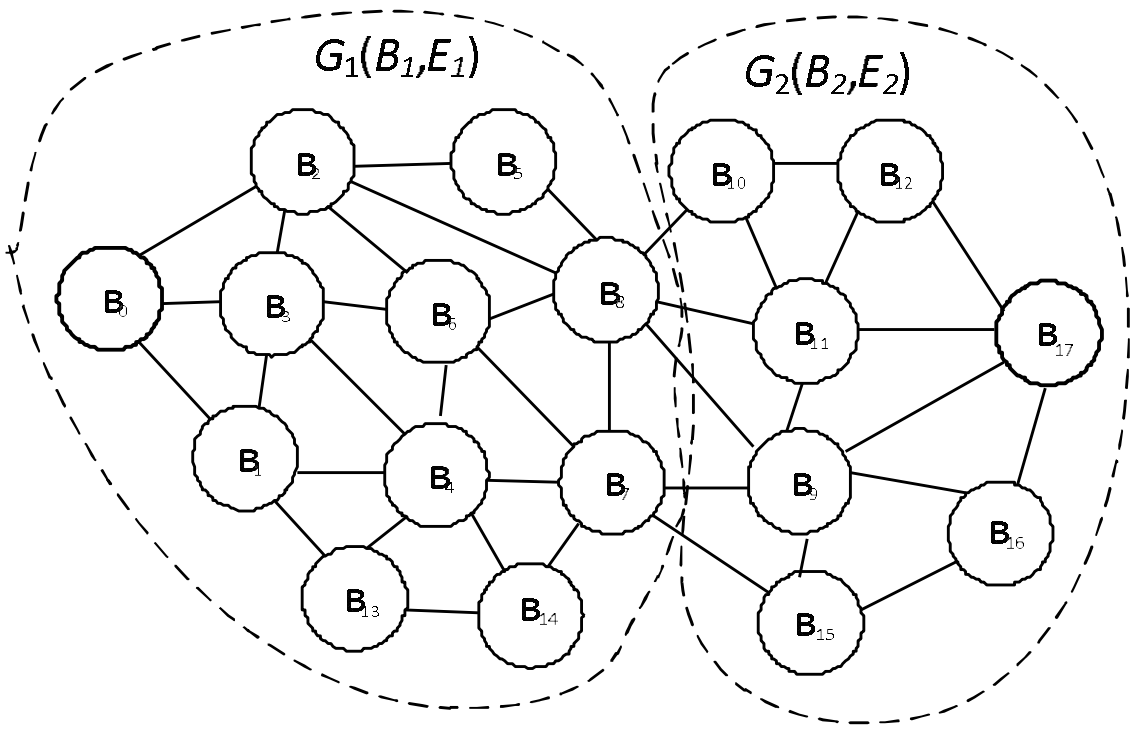


Рис. 2.10 Разделение графа на подграфы

На первом этапе формируются пути от начальной вершины $в_0$ до смежных с ней вершин: $P_1=\{в_0,в_1\}$; $P_2=\{в_0,в_2\}$; $P_3=\{в_0,в_3\}$ (рис. 2.11).

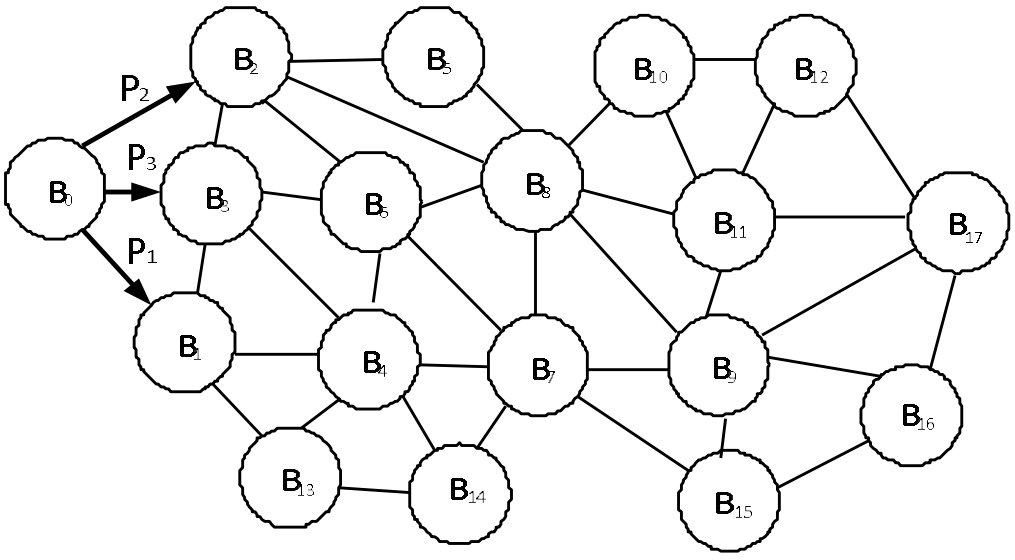


Рис. 2.11 Начальный этап формирования путей

На следующем этапе данные пути продлеваются соответственно до вершин $в_8$, $в_6$, и $в_4$, а именно: $P_1=\{в_0,в_1,в_4\}$; $P_2=\{в_0,в_2,в_8\}$; $P_3=\{в_0,в_3,в_6\}$ (рис.2.12).

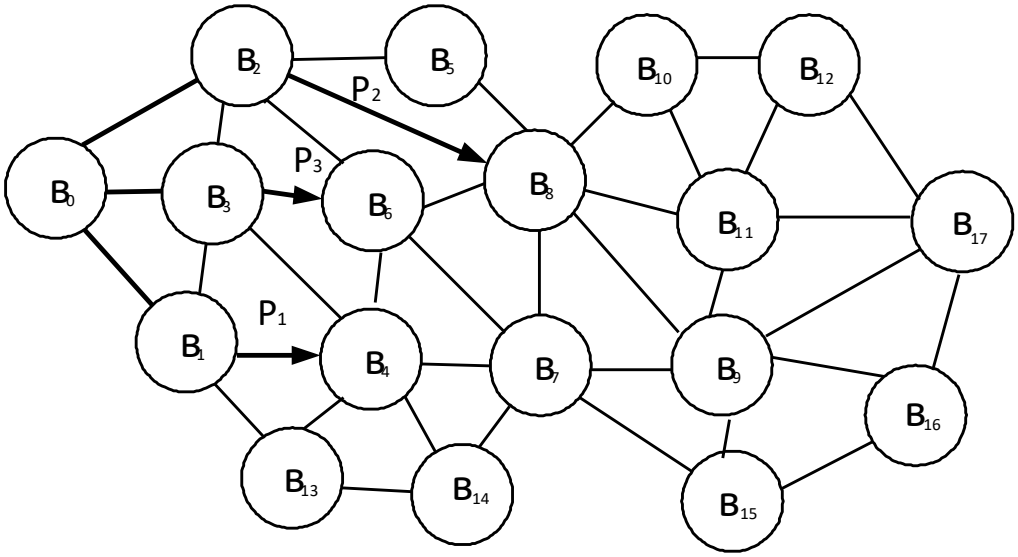


Рис.2.12 Второй этап формирования путей

На данном этапе путь $P_2=\{в_0,в_2,в_8\}$ является сформированным. Затем формируются пути $P_1=\{в_0,в_1,в_4,в_7\}$ и $P_3=\{в_0,в_3,в_6,в_8\}$ (рис.2.13).

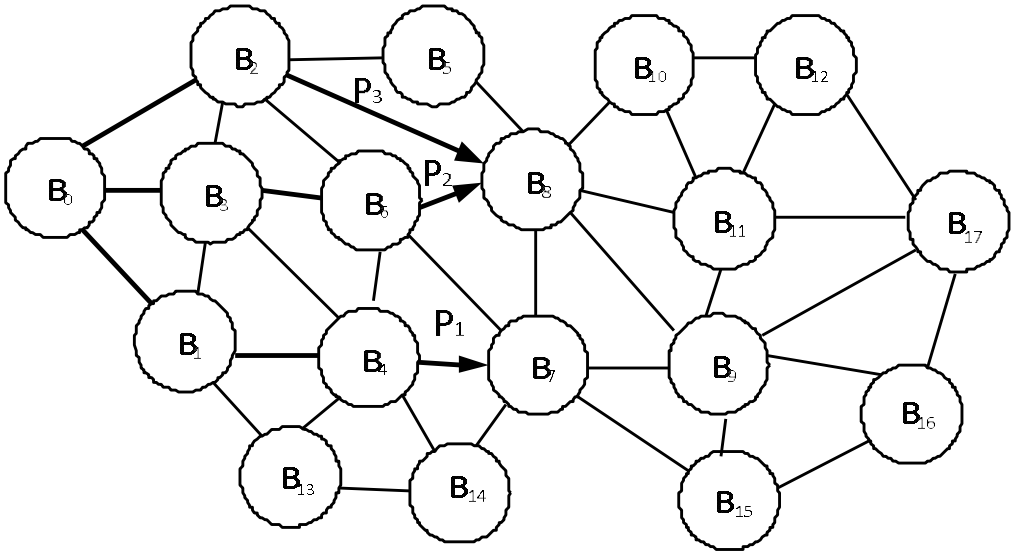


Рис.2.13 Формирование путей до минимального множества сочленения

Формирование всего множества путей между граничными вершинами ($в_7$,$в_8$) и конечной вершиной $в_{17}$ осуществляется аналогичным образом, начиная с конечной вершины $в_{17}$, при этом формируются следующие непересекающиеся пути: $P_4=\{в_8,в_{10},в_{12},в_{17}\}$; $P_5=\{в_8,в_{11},в_{17}\}$; $P_6=\{в_7,в_9,в_{17}\}$; $P_7=\{в_7,в_{15},в_{16},в_{17}\}$ (рис.2.14).

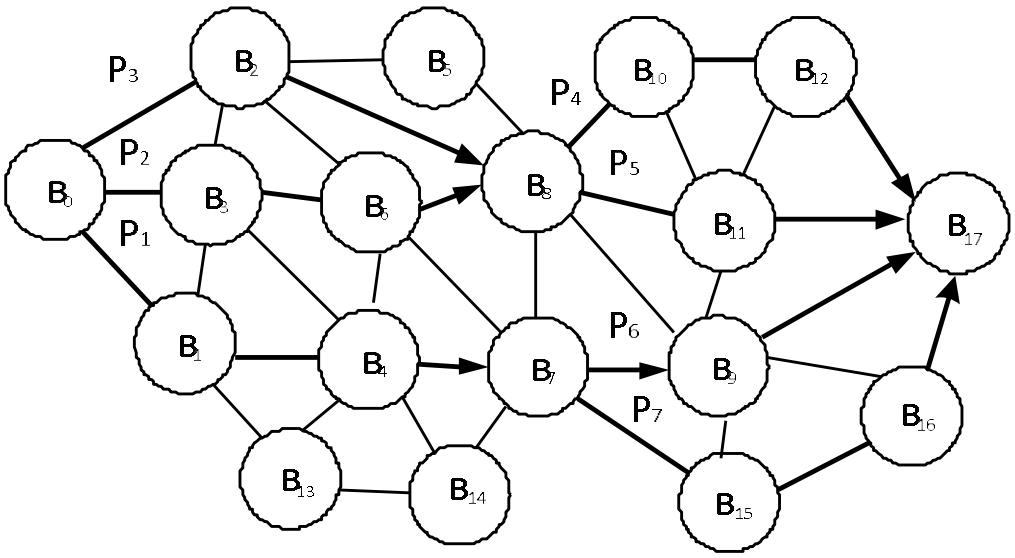


Рис. 2.14 Формирование путей от минимального множества сочленения до конечной вершины

В данном случае между вершинами $в_0$ и $в_{17}$ формируются 6 путей: $P_{16}=\{P_1,P_6\}$; $P_{17}=\{P_1,P_7\}$; $P_{24}=\{P_2,P_4\}$; $P_{25}=\{P_2,P_5\}$; $P_{34}=\{P_3,P_4\}$; $P_{35}=\{P_3,P_5\}$.

В результате между вершинами $в_0$ и $в_{17}$ можно сформировать 8 пар различных непересекающихся путей: (P_{16},P_{24}); (P_{16},P_{25}); (P_{16},P_{34}); (P_{16},P_{35}); (P_{17},P_{24}); (P_{17},P_{25}); (P_{17},P_{34}); (P_{17},P_{35}).

При этом, в зависимости от требуемых параметров QoS между вершинами $в_0$ и $в_{17}$ могут быть сформированы пути различной длины, например путь $P_{25}=\{в_0,в_2,в_8,в_{11},в_{17}\}$ является самым коротким с длиной L_{25}=4. Самыми длинными является пути $P_{17}=\{в_0,в_1,в_4,в_7,в_{15},в_{16},в_{17}\}$ и $P_{34}=\{в_0,в_3,в_6,в_8,в_{10},в_{12},в_{17}\}$ длиной равной 6.

При организации параллельной передачи могут быть использованы пути P_{16} и P_{35} одинаковой длины равной 5. В этом случае параллельно передаваемые части сообщения будут собираться без дополнительной задержки в приемном узле.

На рис.2.15 представлена сборка трех частей сообщения, переданных параллельно по маршрутам с одинаковой задержкой передачи $\tau_i=const$, где: T_i – время передачи i-ой части сообщения, T_d – время добавления (записи) части сообщения.

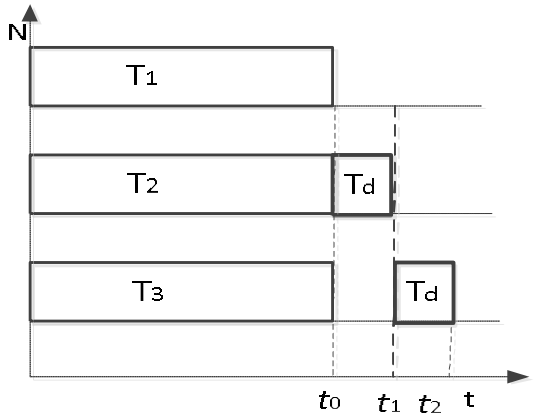


Рис. 2.15 Сборка частей сообщения

В данном случае время сборки всего сообщения $T_{сб}=(t_2-t_0)$ минимально и равно $2T_d$. При задержке передачи каждой части сообщения на величину $\nabla t=(t_1-t_0)$ относительно предыдущей части сообщения время сборки всего сообщения остается минимальным и равно $T_{сб}=(N-1)T_d$.

При задержке $\tau_i>T_d$ передачи i-ой части сообщения время сборки сообщения увеличивается на величину $t_z=\tau_i-T_d$.

Наличие достаточно большого набора всех возможных путей затрудняет процесс конструирования трафика при организации многопутевой передачи информации. В этом случае с целью обеспечения заданных параметров QoS целесообразно формировать многопутевые виртуальные каналы с учетом требований к характеру трафика. Таким образом, возможность формирования различных многопутевых виртуальных каналов позволяет оптимизировать процесс передачи информации в компьютерных сетях большой размерности.

2.5. Способ формирования множества путей от вершины $в_0$ к заданному множеству вершин.

На основании модифицированного метода «ветвей и границ» в диссертационной работе предложен способ одновременного формирования всех непересекающихся путей от вершины $в_0$ к заданному множеству вершин [86].

На начальном этапе формируются пути от исходной вершины до смежных с ней вершин $в_i \in B_s$, которые в данном случае являются граничными для

множества, состоящего из одной вершины $в_0$. Таким образом формируется граф $G_1(B_1,E_1)$ с множеством вершин $B_1=в_0 \cup B_s$. Затем в соответствие с условиями 1-3 граничные вершины включаются в множество внутренних вершин подграфа $G_1(B_1,E_1)$ и формируется новое множество B_s. Таким образом строится дерево решений с корнем в вершине $в_0$ до тех пор пока не сформируются все непересекающиеся пути ко всем заданным вершинам. Временная сложность данного алгоритма равна $O(S_mD)$, где S_m – средняя степень вершин и D – диаметр графа.

Алгоритм формирования множества путей от вершины $в_0$ к заданному подмножеству вершин:

1. Формируем подграф $G_1(B_1,E_1)$ в котором $B_1=\{в_0,B_s\}$, где: $в_0$ – исходная вершина; $B_s=\{в_i|\ i=1,2,\ldots,p\}$ – в данном случае представляет множество вершин, смежных с вершиной $в_0$.
2. Для подграфа $G_1(B_1,E_1)$ определяем множество путей $W=\{L_{o,r}|,r=1,2,\ldots,p\}$, где $L_{o,j}=\{в_0,в_j\}$.
3. Формируем новое множество $B_1=B_1 \cup B_s$.
4. Формируем подграф $G_1(B_1,E_1)$ в котором $B_1=B_1 \cup B_s$.
5. Формируем подграф $G_2(B_2,E_2)$ в котором $B_2=B_2 \setminus B_s$.
6. Для подграфа $G_1(B_1,E_1)$ формируем множество граничных вершин B_s.
7. Если среди вершин $в_i$, смежных с вершиной $в_m$, есть вершина $в_k \in B_z$, то формируем путь к вершине $в_k$.
8. Среди множества вершин $B_s=\{в_i|\ i=1,2,\ldots,p\}$ выбираем вершину $в_i$ с минимальной внешней степенью $S^o_k=\min\{S^o_i|\ i=1,2,\ldots,p\}$.
9. Для вершины $в_i$ находим внутреннюю вершину $в_j \in B^i_2$ подграфа $G_2(B_2,E_2)$ с минимальной степенью.
10. Продлеваем путь $L_{o,i}$ до вершины $в_j \in B^i_2$.
11. Если $в_j \notin B^V$ то ($в_j \subset B_c$, $в_i \subset B^i_1$) иначе (путь $L_{o,j}$ сформирован).
12. Если множество $B_1 \neq \varnothing$ то (переход к пункту 7).
13. Конец.

На рис.2.16 представлено значение временной сложности формирования 5 путей от вершины $в_1$ до вершины $в_{10}$ для графа, состоящего из 28 узлов и степенью вершин S_1 и S_{10} равной 5.

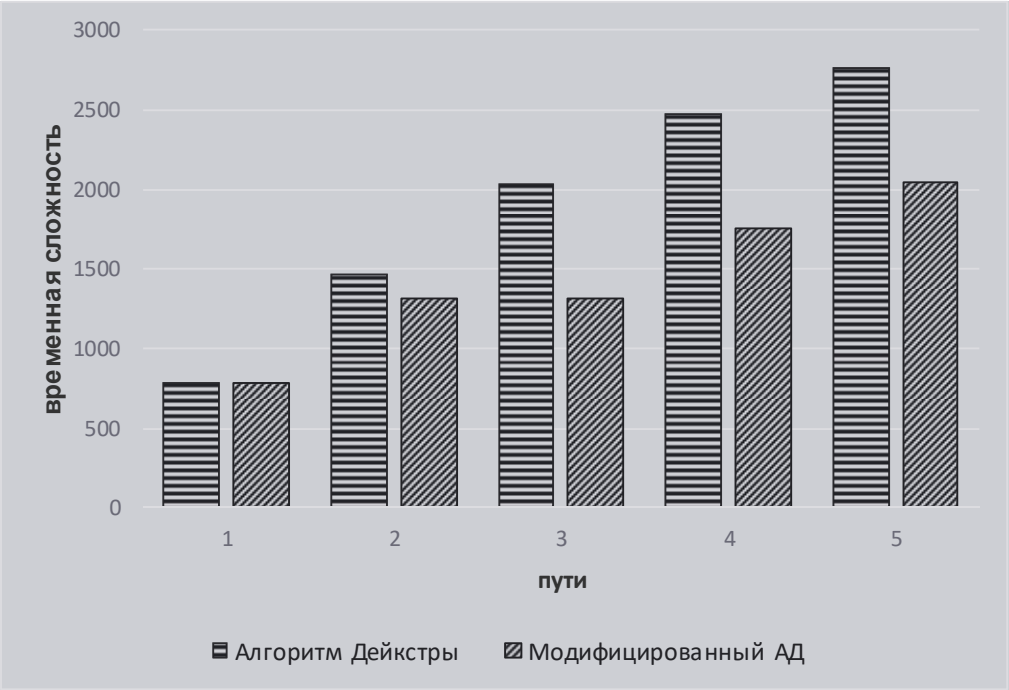


Рис. 2.16 Временная сложность формирования 5 путей от вершины $в_1$ вершины $в_{10}$

Как видно из рис. 2.16 временная сложность формирования каждого последующего пути при модифицированном алгоритме меньше, чем при базовом алгоритме.

На рис. 2.17 представлено значения временной сложности формирования 4 путей от вершины $в_1$ до вершины $в_{11}$ со степенью S_{11}=4.

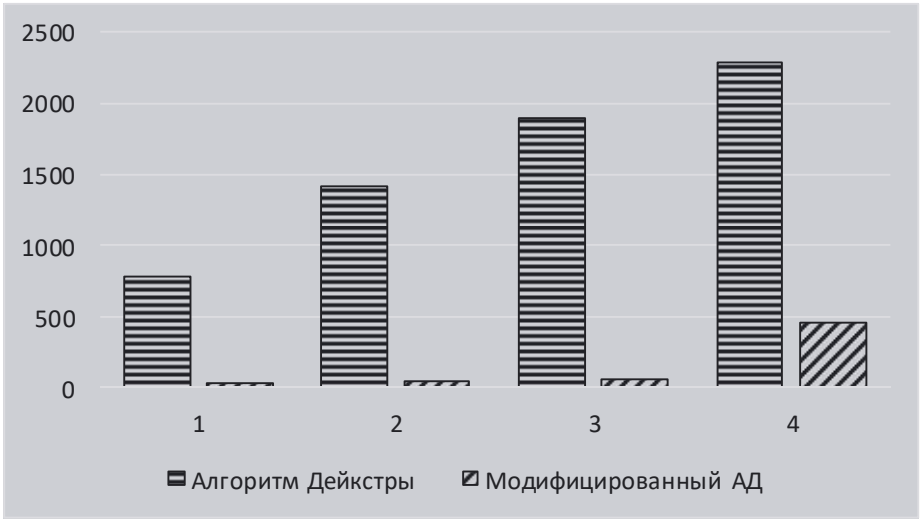


Рис. 2.17 Временная сложность формирования 4 путей от вершины $в_1$ до вершины $в_{11}$

Как видно из рис. 2.17 временная сложность базового алгоритма формирования второго пути осталась практически такой же как и при формировании первого пути. При модифицированном алгоритме осуществляется одновременное формирование множества путей от одного узла к нескольким. В этом случае при формировании второго и последующих путей используются пути сформированные на предудущей итерации.

2.6. Поиск множества путей к нескольким вершинам с учетом метрики каждого пути.

Поиск кратчайших путей от вершины $в_1$ к вершинам $в_{10}$, $в_{11}$, и $в_{26}$. Для каждого пути P_i задана метрика M_i равная сумме весов всех звеньев пути.

На рис 2.18 представлен результат формирования множества путей от $в_1$ к ближайшей вершине $в_{10}$ с учетом метрики путей.

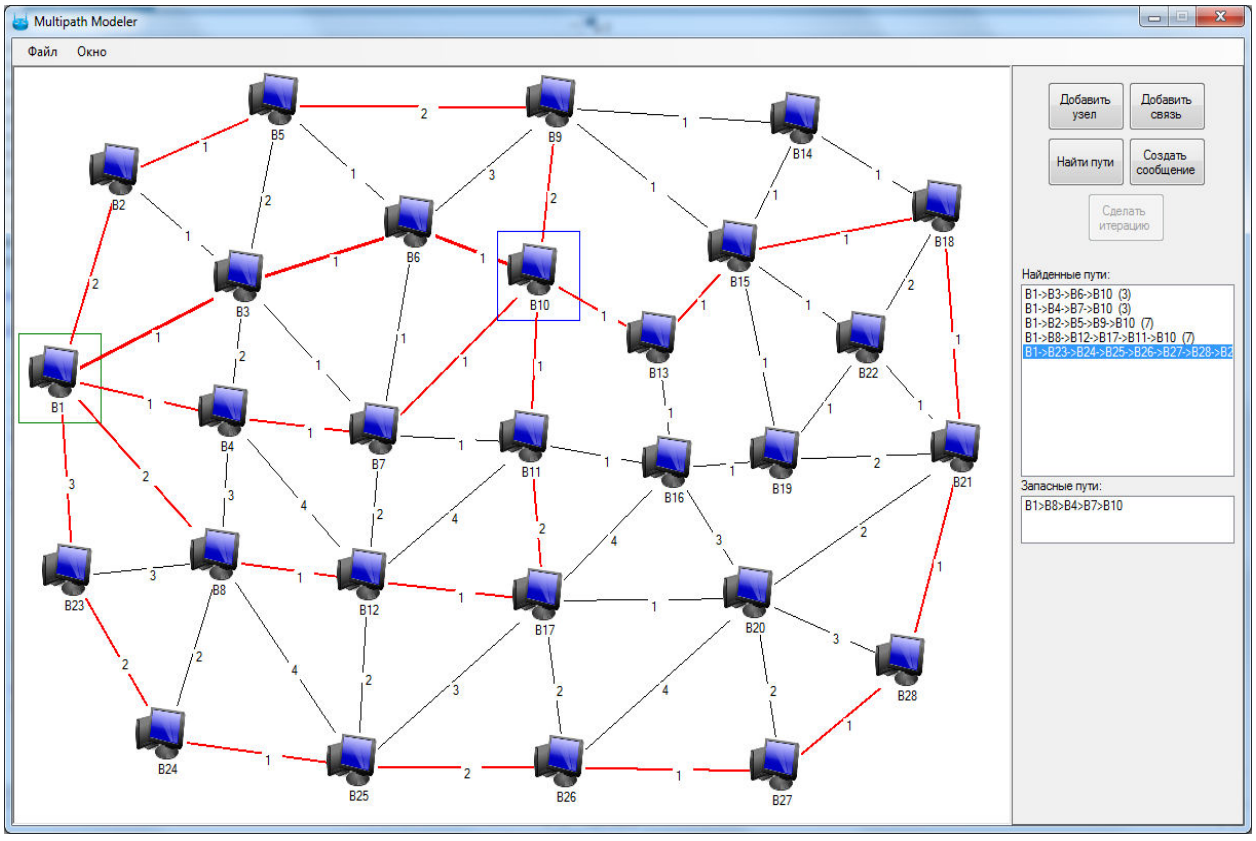


Рис. 2.18 Формирование множества путей от $в_1$ к ближайшей вершине $в_{10}$

В результате формируются пути:

$P_1=\{в_1,в_3,в_6,в_{10}\}$ с метрикой $M_1=3$;

$P_2=\{в_1,в_4,в_7,в_{10}\}$ с метрикой $M_2=3$;

$P_3=\{в_1,в_2,в_5,в_9,в_{10}\}$ с метрикой $M_3=7$;

$P_4=\{в_1,в_8,в_{12},в_{17},в_{11},в_{10}\}$ с метрикой $M_4=7$;

$P_5=\{в_1,в_{23},в_{24},в_{25},в_{26},в_{27},в_{28},в_{21},в_{18},в_{15},в_{13},в_{10}\}$ с метрикой $M_5=15$.

Одновременно с этим формируется путь $P_6=\{в_1,в_8,в_{12},в_{17},в_{11}\}$ к вершине $в_{11}$ с метрикой $M_6=6$.

После этого продолжает формироваться дерево путей до следующей ближайшей вершины $в_{11}$. Первым формируется путь $P_7=\{в_1,в_3,в_6,в_{10},в_{11}\}$ с метрикой $M_6=4$, который является продолжением пути $P_1=\{в_1,в_3,в_6,в_{10}\}$ с метрикой $M_1=3$ (рис. 2.19).

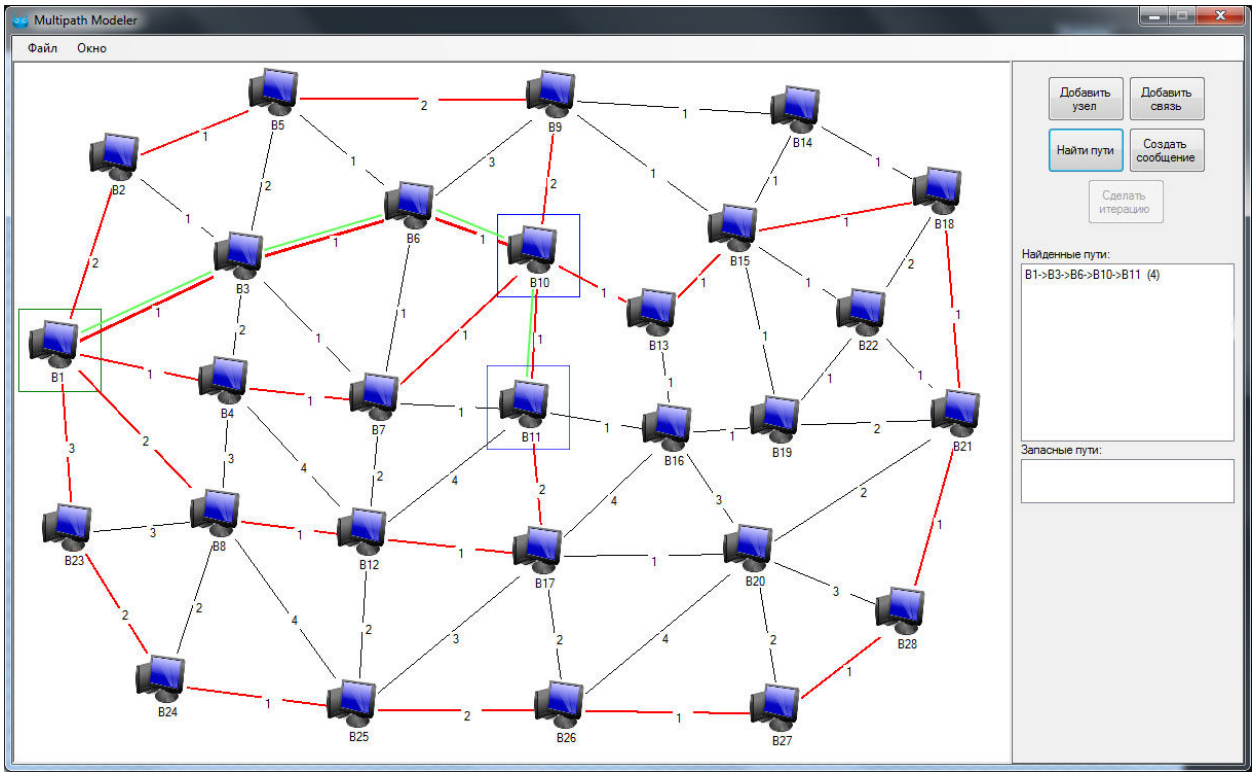


Рис. 2.19 Продление пути $P_1=\{в_1,в_3,в_6,в_{10}\}$ до вершины $в_{11}$

Затем осуществляется формирование остальных путей к вершине $в_{11}$:

$P_8=\{в_1,в_4,в_7,в_{11}\}$ с метрикой $M_2=3$;

$P_9=\{в_1,в_2,в_5,в_9,в_{15},в_{13},в_{16},в_{11}\}$ с метрикой $M_9=9$;

В результате формируются следующие пути к вершине $в_{11}$ (рис. 2.20):

$P_6=\{в_1,в_8,в_{12},в_{17},в_{11}\}$;

$P_7=\{в_1,в_3,в_6,в_{10},в_{11}\}$;

P_8={$в_1$,$в_4$,$в_7$,$в_{11}$};

P_9={$в_1$,$в_2$,$в_5$,$в_9$,$в_{15}$,$в_{13}$,$в_{16}$,$в_{11}$}.

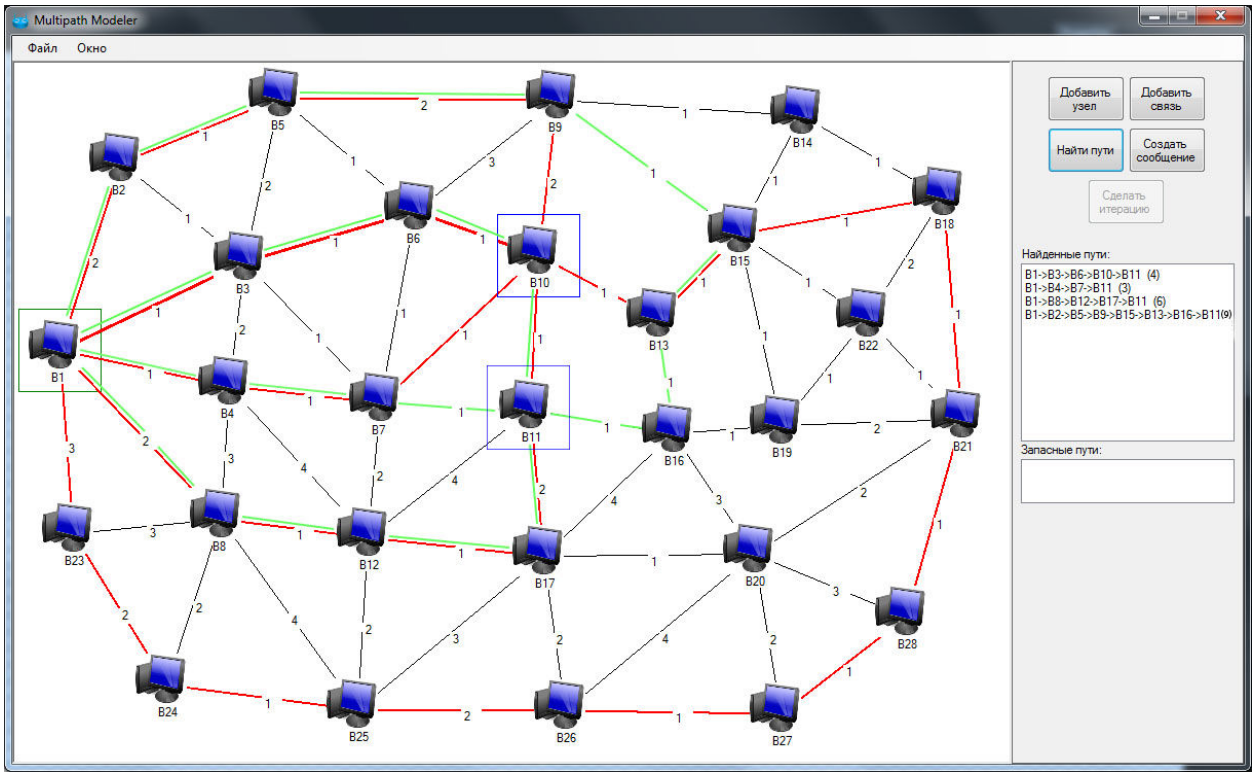


Рис. 2.20 Формирование путей к вершине $в_{11}$

Формирование путей до вершины $в_{26}$ происходит одновременно с формированием путей к другим вершинам, так путь P_{10}={$в_1$,$в_{23}$,$в_{24}$,$в_{25}$,$в_{26}$} с метрикой M_{10}=6 является частью пути P_5. Путь P_{11}={$в_1$,$в_8$,$в_{12}$,$в_{17}$,$в_{26}$} с метрикой M_{11}=6 является разветвлением пути P_6={$в_1$,$в_8$,$в_{12}$,$в_{17}$,$в_{11}$} в вершине $в_{17}$. Путь P_{12}={$в_1$,$в_4$,$в_7$,$в_{11}$,$в_{16}$,$в_{20}$,$в_{27}$,$в_{26}$} с метрикой M_{12}=10 является продолжением пути из вершины $в_{11}$ до вершины $в_{26}$ (рис. 2.21).

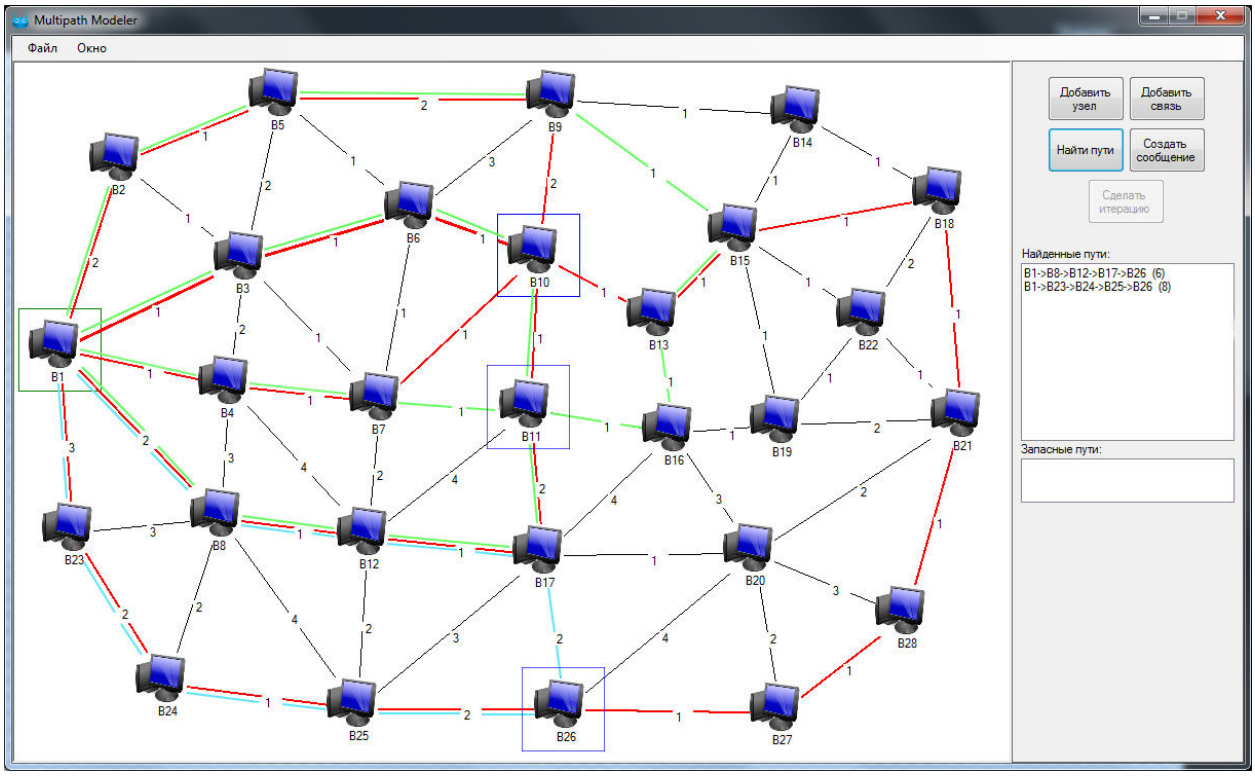
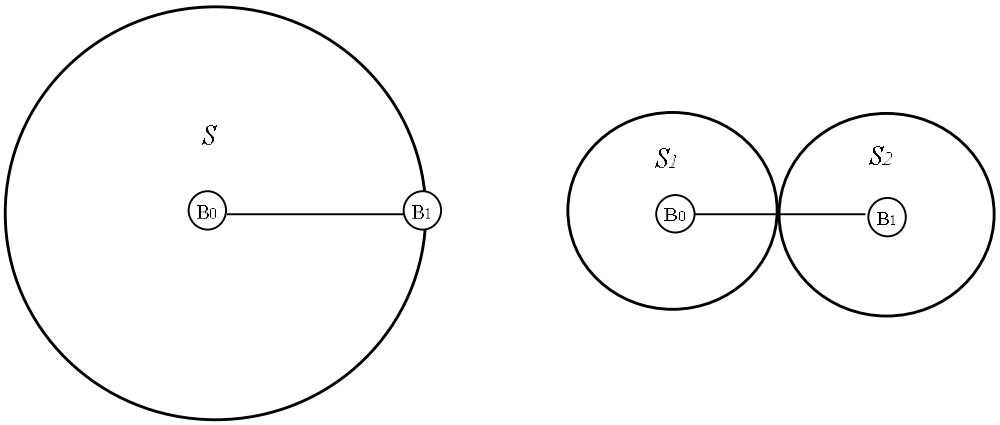


Рис. 2.21 Формирование путей к вершине $в_{26}$

2.7. Формирование множества непересекающихся путей на основе метода «встречных потоков»

Основным недостатком волновых алгоритмов и алгоритмов поиска в глубину или ширину, является достаточно большая область поиска решений. Так, при регулярной структуре сети большой размерности область поиска первого кратчайшего пути от начальной вершины $в_0$ к вершине $в_k$ (рис. 2.18а) равна $S=\pi L^2$, где L – минимальное расстояние между вершинами $в_0$ и $в_к$.

В работе [87] предложен способ формирования множества путей между двумя вершинами, позволяющий уменьшить область поиска за счет одновременного формирования деревьев путей из начальной вершины $в_0$ и конечной вершины $в_к$. В этом случае (рис. 2.22б) $S=S_1+S_2$, где $S_1=S_2=\pi L^2/4$. Отсюда $S_1+S_2=\pi L^2/2$. Таким образом, область поиска решений уменьшается в два раза, соответственно, и время формирования пути уменьшается в два раза.



а) поиск пути от $в_0$ к $в_к$ б) встречный поиск путей между $в_0$ и $в_к$

Рис. 2.22. Область поиска путей между $в_0$ и $в_к$

Многопутевая маршрутизация на основе встречных потоков сводится к нахождению точек соединения деревьев, сформированных с помощью волновых алгоритмов или модифицированного алгоритма «ветвей и границ».

Необходимым и достаточным условием существования пути между корневыми вершинами деревьев является наличие точек их соединения. Наличие точек соединения определяется следующим условием.

Лемма 3. Деревья $T_i(B_i,E_i)$ и $T_i(B_i,E_i)$ имеют точки соединения при $B_i \cap B_j \neq \varnothing$ и $E_i \cap E_j = \varnothing$.

Вершины $E_0=E_i \cap E_j$ являются точками соединения деревьев $T_i(B_i,E_i)$ и $T_i(B_i,E_i)$.

Таким образом, поиск путей между вершинами деревьев может быть сведен к задаче нахождения множества точек их соединения. При этом алгоритм формирования деревьев может быть любой в зависимости от постановки задачи и требованиям к маршрутам передачи информации.

Вершина $в_0$ относится к множеству B_0 внутренних вершин подграфа $G_1=(B_1,E_1)$. При формировании путей в множество B_0 каждый раз добавляется вершина $в_i \in B_b$, имеющая меньшее число внешних ветвей по сравнению с остальными граничными вершинами.

Соответственно, в множество B_b добавляется вершина $в_j$, смежная с вершиной $в_i$ с минимальной внешней степенью S^b_j. Таким образом, строится дерево решений с корнем в вершине $в_0$ пока не сформируются все непересекающиеся пути к заданным вершинам.

На рис. 2.23 представлена блок схема алгоритма формирования множества непересекающихся путей с применением методом «встречных потоков».

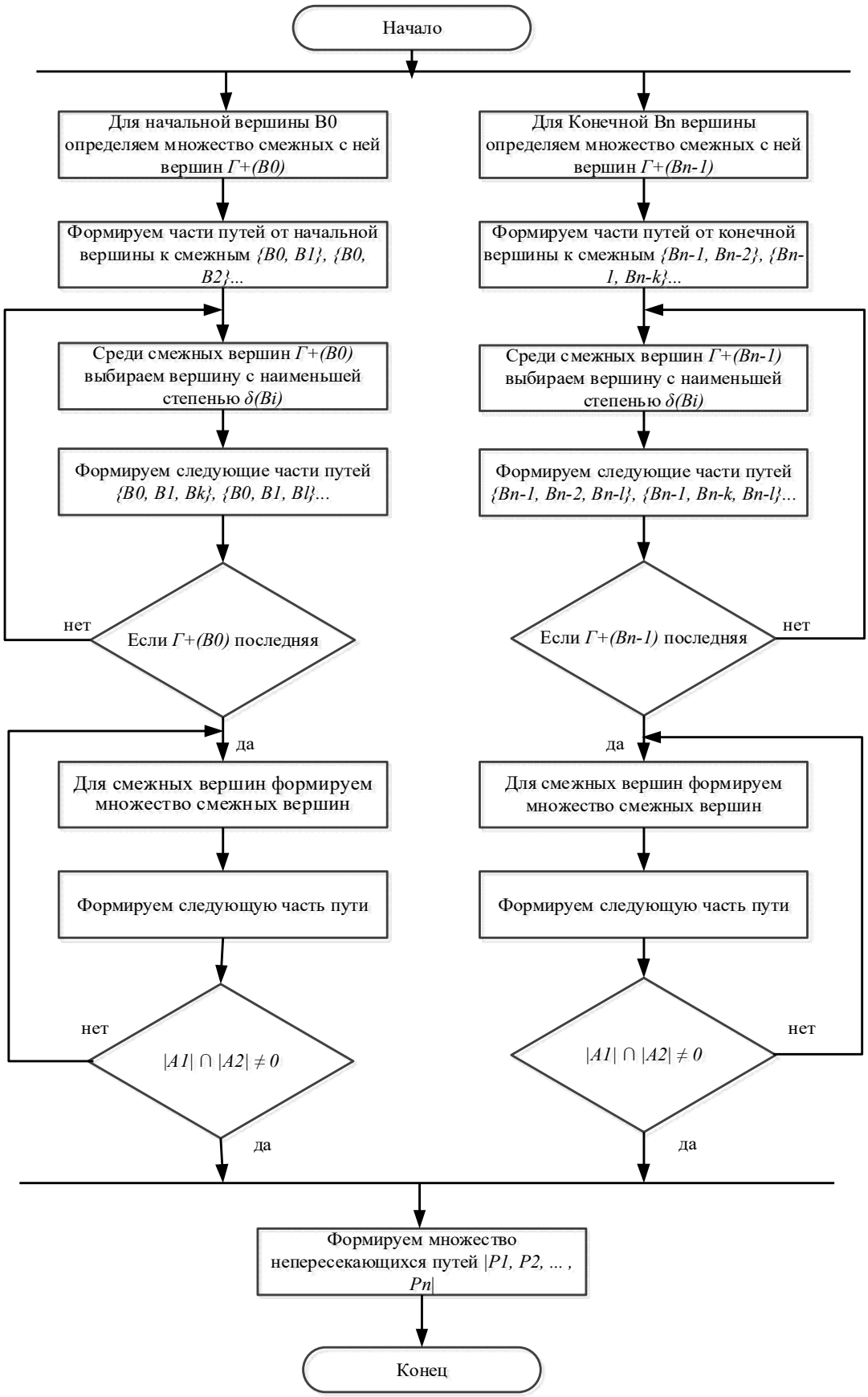


Рис. 2.23. Алгоритм формирования множества непересекающихся путей

Рассмотрим пример формирования множества путей для графа, представленного на рис. 2.24.

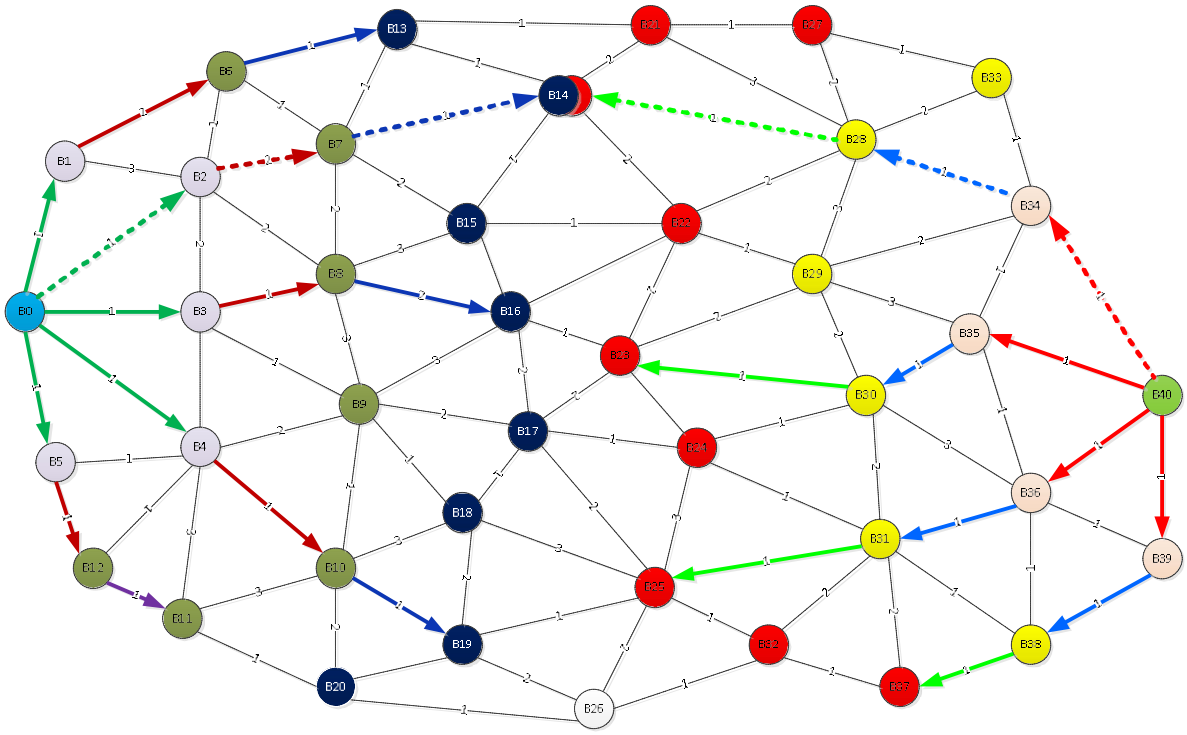


Рис. 2.24. Формирование первого непересекающегося пути

На первом этапе формируются пути от начальной вершины B_0 до смежных с ней вершин: $P_{н1}=\{в_0,в_1\}$; $P_{н2}=\{в_0,в_2\}$; $P_{н3}=\{в_0,в_3\}$; $P_{н4}=\{в_0,в_4\}$; $P_{н5}=\{в_0,в_5\}$. На следующем этапе формируются пути от конечной вершины $в_{40}$ до смежных с ней вершин: $P_{к1}=\{в_{40},в_{39}\}$; $P_{к2}=\{в_{40},в_{36}\}$; $P_{к3}=\{в_{40},в_{35}\}$; $P_{к4}=\{в_{40},в_{34}\}$.

Далее формируется следующее множество граничных вершин относительно внутренних смежных от начальной вершины $\{в_6,в_7,в_8,в_9,в_{10},в_{12}\}$. Среди внутренних смежных вершин выбирается вершина с наименьшей степенью и формируется часть пути. Так происходит до тех пор, пока от каждой внутренней смежной вершины не будет сформирована часть пути к граничным вершинам:

$P_{н1}=\{в_0\text{-}в_1\text{-}в_6\}$;

$P_{н2}=\{в_0\text{-}в_2\text{-}в_7\}$;

$P_{н3}=\{в_0\text{-}в_3\text{-}в_8\}$;

$P_{н4}=\{в_0\text{-}в_4\text{-}в_{10}\}$;

$P_{н5}=\{в_0\text{-}в_5\text{-}в_{12}\}$.

Аналогичная процедура формирования путей происходит параллельно и от конечной вершины.

$P_{к1}=\{в_{40}\text{-}в_{39}\text{-}в_{38}\}$;

$P_{к2}=\{в_{40}\text{-}в_{36}\text{-}в_{31}\}$;

$P_{к3}=\{в_{40}\text{-}в_{35}\text{-}в_{30}\}$;

$P_{к4}=\{в_{40}\text{-}в_{34}\text{-}в_{28}\}$.

На третьем такте происходит пересечение двух путей $P_{н2}\cap P_{к4}$, в вершине B_{14}, в результате которого сформирован первый непересекающийся путь: $P_1=\{в_0\text{-}в_2\text{-}в_7\text{-}в_{14}\text{-}в_{28}\text{-}в_{34}\text{-}в_{40}\}$. Формирование всего множества путей между граничными вершинами от начала и конца осуществляется аналогичным образом, при этом формируются следующие непересекающиеся пути:

$P_2=\{в_0\text{-}в_3\text{-}в_8\text{-}\underline{в_{16}}\text{-}в_{23}\text{-}в_{30}\text{-}в_{35}\text{-}в_{40}\}$

$P_3=\{в_0\text{-}в_4\text{-}в_{10}\text{-}в_{19}\text{-}\underline{в_{25}}\text{-}в_{31}\text{-}в_{36}\text{-}в_{40}\}$

$P_4=\{в_0\text{-}в_5\text{-}в_{12}\text{-}в_{11}\text{-}в_{20}\text{-}\underline{в_{26}}\text{-}в_{32}\text{-}в_{37}\text{-}в_{38}\text{-}в_{39}\text{-}в_{40}\}$

В таблицах 2.1 и 2.2 представлено формирования путей по тактам.

Таблица 2.1.

Формирования путей от начальной вершины

t_1	B_0-B_1	B_0-B_2	B_0-B_3	B_0-B_4	B_0-B_5
t_2	B_0-B_1-B_6	B_0-B_2-B_7	B_0-B_3-B_8	B_0-B_4-B_{10}	B_0-B_5-B_{12}
t_3	B_0-B_1-B_6-B_{13}	B_0-B_2-B_7-B_{14}	B_0-B_3-B_8-B_{16}	B_0-B_4-B_{10}-B_{19}	B_0-B_5-B_{12}-B_{11}
t_4	B_0-B_1-B_6-B_{13}-B_{21}	-	B_0-B_3-B_8-B_{16}--B_{23}	B_0-B_4-B_{10}-B_{19}--B_{25}	B_0-B_5-B_{12}-B_{11}--B_{20}
t_5	B_0-B_1-B_6-B_{13}-B_{21}- -B_{27}	-	-	-	B_0-B_5-B_{12}-B_{11}-B_{20}-B_{26}
t_6	B_0-B_1-B_6-B_{13}-B_{21}- -B_{27}-B_{33}	-	-	-	-
t_7	Путь не сформирован!	-	-	-	-

Таблица 2.2

Формирования путей от конечной вершины

t_1	B_{40}-B_{39}	B_{40}-B_{36}	B_{40}-B_{35}	B_{40}-B_{34}
t_2	B_{40}-B_{39}-B_{38}	B_{40}-B_{36}-B_{31}	B_{40}-B_{35}-B_{30}	B_{40}-B_{34}-B_{28}
t_3	B_{40}-B_{39}-B_{38}-B_{37}	B_{40}-B_{36}-B_{31}-B_{25}	B_{40}-B_{35}-B_{30}-B_{23}	B_{40}-B_{34}-B_{28}-B_{14}
t_4	B_{40}-B_{39}-B_{38}-B_{37}-B_{32}	-	-	-
t_5	B_{40}-B_{39}-B_{38}-B_{37}-B_{32}-B_{26}	-	-	-

Из рисунка 2.25 видно, что путь $P_{н1}=\{B_0\text{-}B_1\text{-}B_6\text{-}B_{13}\text{-}B_{21}\text{-}B_{27}\text{-}B_{33}\}$ продолжает поиск граничных вершин относительно начала и формировать часть пути. На такте t_7, граничной вершиной относительно к внутренней смежной является вершина B_{34}, которая уже входит в сформированный путь $P_1=\{B_0\text{-}B_2\text{-}B_7\text{-}\underline{B_{14}}\text{-}B_{28}\text{-}B_{34}\text{-}B_{40}\}$. В связи с этим, путь $P_{н1}=\{B_0\text{-}B_1\text{-}B_6\text{-}B_{13}\text{-}B_{21}\text{-}B_{27}\text{-}B_{33}\}$ отбрасывается. В результате сформировано 4 не пересекающихся пути.

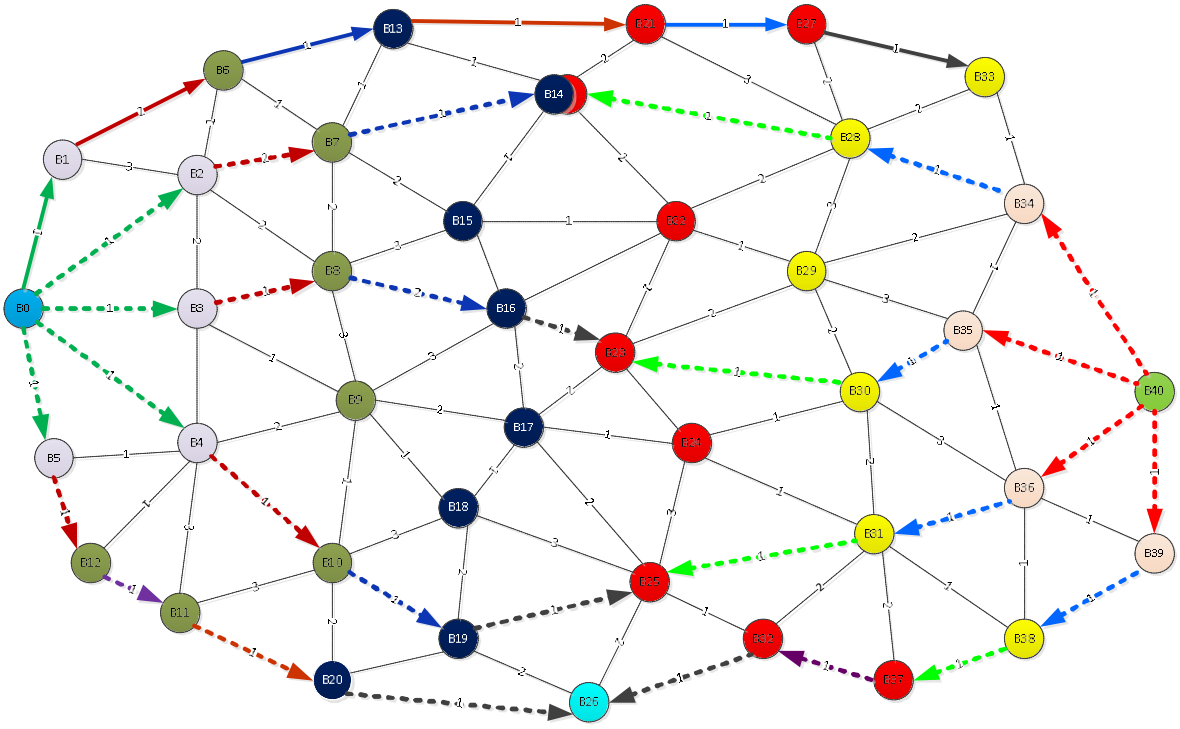


Рис. 2.25. Формирование множества непересекающихся путей

ВЫВОДЫ ПО ГЛАВЕ 2

1. Предложенный в работе модифицированный алгоритм «ветвей и границ» за счет одновременного формирования множества путей от одного узла ко всем остальным узлам существенным образом уменьшает временную сложность формирования виртуальной структуры Grid-систем.

2. Был проведен сравнительный анализ эффективности использования предложенного волнового алгоритма маршрутизации по сравнению с классическим и модифицированным алгоритмами Дейкстры.

3. Показано, что предложенный алгоритм позволяет существенным образом сократить временную сложность формирования множества непересекающихся путей сети, что является существенным преимуществом сравнительно с существующими способами.

4. Предложен способ, позволяющий при использовании волнового алгоритма сформировать множество непересекающихся путей, незначительно отличающихся по своей длине.

5. Предложенный в работе способ многопутевой маршрутизации основан на определении множества вершин сочленения деревьев с корневыми вершинами двух деревьев, между которыми формируются непересекающиеся пути. За счет одновременного формирования этих деревьев сокращается время и область поиска множества непересекающихся путей.

ГЛАВА 3

ОРГАНИЗАЦИЯ МНОГОПУТЕВЫХ ВИРТУАЛЬНЫХ КАНАЛОВ В СЕТИ MPLS

3.1. Формирование множества непересекающихся путей между граничными маршрутизаторами сети MPLS

Альтернативным способом достигнуть оптимального TE является подход, основанный на MPLS [88]. В MPLS, label-switched paths (LSPs) настроены использовать сигнальный протокол для отправки пакета Resource ReSLERVation Protocol (RSVP) [89].

Маршрутизация в сети MPLS осуществляется с помощью метода коммутации по меткам. С этой целью во внутренних узлах сети MPLS помещаются маршрутизаторы *LSR* (Label-Switched Router), осуществляющие коммутацию по меткам. Каждый маршрутизатор LSR_i на основании содержимого своей таблицы меток и входящей метки пакета присваивает ему исходящую метку и передает пакет на исходящий интерфейс. Процесс формирование таблиц меток осуществляется с помощью протокола распределения меток LDP [90] путем обмена информацией между смежными маршрутизаторами сигнальными пакетами запрос метки *RL* (рис. 3.1) и отображение метки *ML* (рис. 3.2).

Запрос метки (0×0401)	Длина сообщения
Идентификатор сообщения	
Элемент FEC	

Рис. 3.1. Формат пакета запрос метки RL протокола LDP

Отображение метки (0×0400)	Длина сообщения
Идентификатор сообщения	
Элемент FEC	
Метка	

Рис. 3.2. Формат пакета отображения метки ML протокола LDP

Элемент *FEC* меток содержит адрес начального маршрутизатора $A(LER_O)$, адрес конечного маршрутизатора $A(LER_I)$, номер пути между маршрутизаторами LER_O и LER_I и метрику пути. В простейшем случае в качестве метрики может выступать количество каналов на пути между маршрутизаторами LER_O и LER_I.

В работе [91] представлен алгоритм формирования таблиц меток для организации многопутевой маршрутизации в сети MPLS, который основан на алгоритме формирования множества непересекающихся путей, приведенного в разделе 2.2. настоящей работе.

Формирование таблиц меток связано с построением дерева путей $T(V_T,E_T)$, осуществляется граничным маршрутизатором с минимальным числом внешних связей. При этом путь продлевается до смежного маршрутизатора с минимальным числом связей с вершинами дерева путей $T(V_T,E_T)$. Это обеспечивает возможность построения максимального числа непересекающихся путей.

3.2. Алгоритм формирования таблиц меток между начальным LERo и конечным LERn маршрутизаторами.

1. Начало
2. Из маршрутизаторов, смежных с маршрутизатором LER_o, формируется множество граничных маршрутизаторов *MBR*.
3. В результате обмена пакетами *RL* и *ML* формируются таблицы меток маршрутизатора LER_o и маршрутизаторов $LSR_i \in MBR$.
4. Определяется маршрутизатор $LSR^m := LSR_i \in MBR$ с минимальным числом внешних связей.
5. Среди маршрутизаторов $LSR_i \notin V_T$ смежных с маршрутизатором LSR^m выбирается внешний смежный маршрутизатор $LSR^v := LSR_i \notin V_T$ с минимальным числом внешних связей.
6. В результате обмена пакетами *RL* и *ML* формируются таблицы меток маршрутизатора $LSR^m := LSR_i \in MBR$ и маршрутизатора LSR^v.

7. Если маршрутизатор LSR^v не является маршрутизатором LER_n, то путь не сформирован, переход к пункту 9.

8. Пересылка пакета подтверждение маршрута *CR* в обратном направлении и корректировка таблиц меток на пути от LER_n к LER_o

9. Если степень внешнего маршрутизатора равна 1, то переход к п. 12.

10. Граничный маршрутизатор LSR^m становится внутренним для $T(V_T,E_T)$.

11. Внешний маршрутизатор LSR^v становится граничным для $T(V_T,E_T)$.

12. Если множество граничных маршрутизаторов не пустое, то переход к п. 3.

13. Конец.

На каждом шаге построения дерева путей, представленного в виде подграфа $G_1(V_1,E_1)$ исходного графа $G_0(V_0,E_0)$ формируются множества внутренних и граничных вершин подграфа $G_1(B_1,E_1)$. Множество вершин $V_1=V_E\cup V_S$, где V_E – множество граничных вершин, а V_S – множество внутренних вершин подграфа $G_1(V_1,E_1)$. Ребра, связывающие внутренние вершины, также являются внутренними ребрами, то есть ребро *e*. Вершины множества $V_E=\{v^e_i|\ i=1,2,\ldots,n\}$, смежные с вершинами множества $V_2=V_0\backslash V_1$, являются граничными вершинами подграфа $G_1(V_1,E_1)$. Маршрутизаторы LSR_j, соответствующие вершинам множества $V_E=\{v^e_i|\ i=1,2,\ldots,n\}$ образуют множество *MBR* граничных маршрутизаторов.

Вершины множества $V_S=\{v^s_i\ \ i=1,2,\ldots,n\}$ подграфа $G_1(V_1,E_1)$ не смежные с вершинами множества $V_2=V_0\backslash V_1$ будем относить к внутренним вершинам. Для множества внутренних вершин $V_S\subset V_1$ выполняется условие $E_s=\{e^i_{k,j}|\ в_k\in B_1,\ в_j\in\ B_1\}$. Соответственно ребро $e^i_{k,j}$ является внутренним ребром. Каждой внутренней вершине $в^s_j\in BS_1$ соответствует маршрутизатор LER_j или LSR_j, совокупность которых образует множество *MLSR* внутренних маршрутизаторов и множество *MBR* граничных маршрутизаторов. В результате последовательного формирования дерева путей формируется множество путей между

маршрутизаторами LER_O и LER_I. При этом на каждом шаге формирования дерева решений путь продлевается относительно граничного маршрутизатора дерева решений в качестве источника пакета «запрос метки» выступает граничный маршрутизатор подсети с минимальным числом внешних связей. В качестве приемника пакета «запрос метки» выступает внешний смежный маршрутизатор с минимальным числом внешних связей.

Особенностью данного алгоритма является то, что путь формируется в обратном направлении от получателя к отправителю. При этом формирование осуществляется децентрализованным способом. Таблицы меток формируются путем обмена метками между смежными маршрутизаторами. Это позволяет не хранить всю маршрутную информацию в каждом маршрутизаторе.

3.3. Пример формирования путей между граничными маршрутизаторами

В качестве примера рассмотрим процесс формирования путей между граничными маршрутизаторами LER_1 и LER_4 (рис. 3.3).

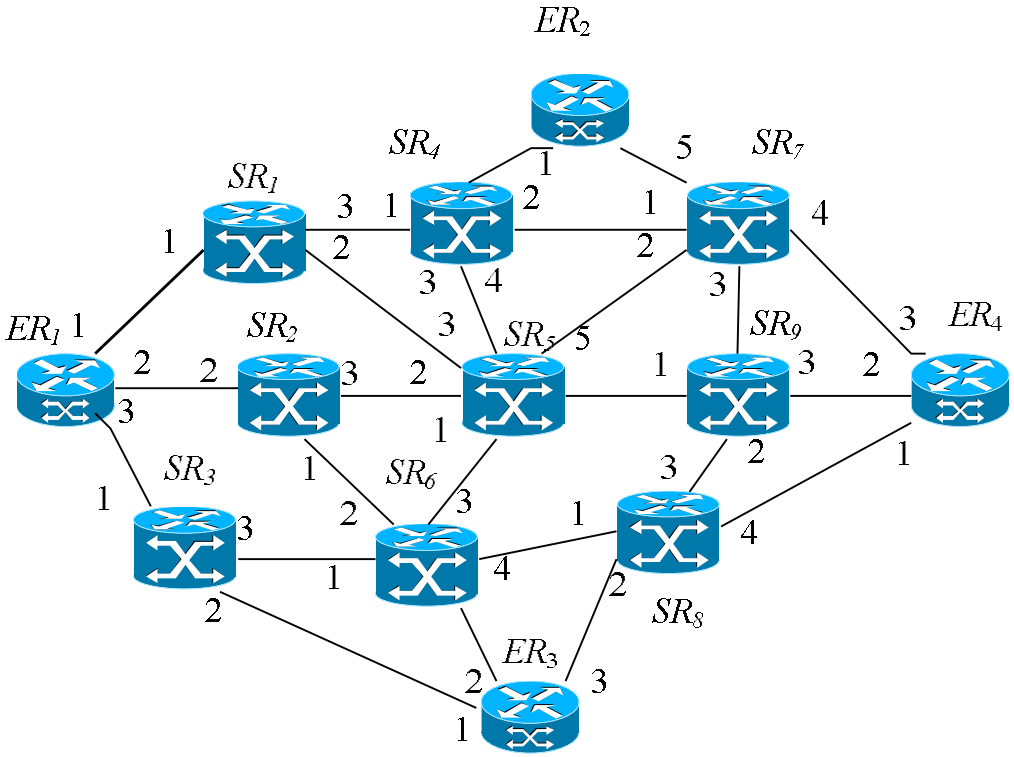


Рис. 3.3. Граф сети MPLS

На рис. 3.4 приведен пример последовательности формирования таблиц меток при организации пути между граничными маршрутизаторами LER_1 и LER_4, где: *RL* – запрос метки; *ML* – отображение метки *CR* – подтверждение маршрута; 1 – формирования таблицы меток; 2 – выбор следующего маршрутизатора; 3 – формирование обратного пути; 4 – формирования таблиц меток обратного пути; 5 – завершение формирования маршрута.

Процесс формирования множества путей между граничным маршрутизатором LER_1 с адресом A(LER_1) и граничным маршрутизатором LER_4 с адресом A(LER_4) осуществляется следующим образом.

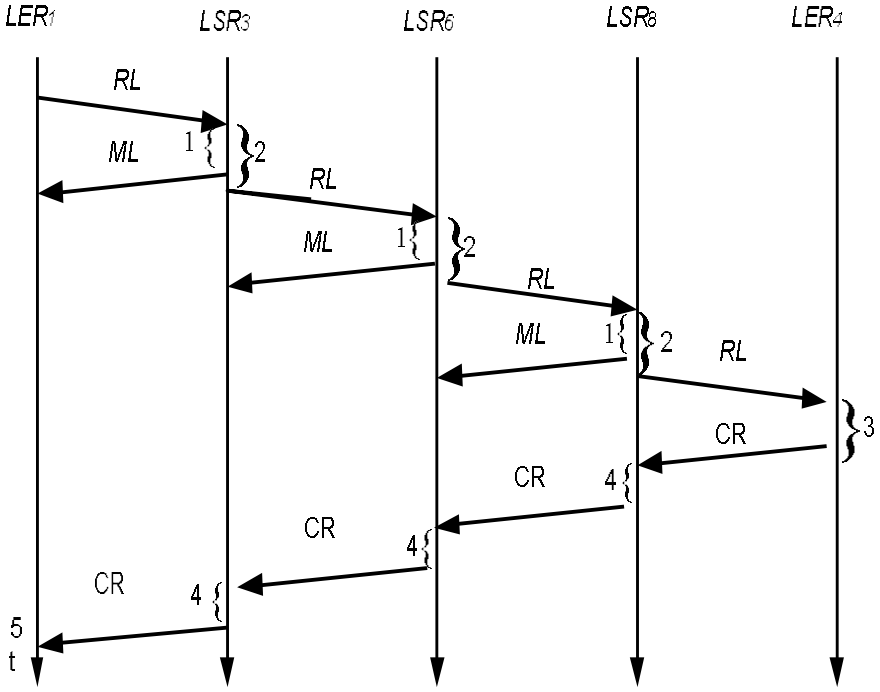


Рис. 3.4. Временная диаграмма формирования таблиц меток между маршрутизаторами LER_1 и LER_4

На начальном этапе формируются таблицы меток для множества сетевых маршрутизаторов BR={LSR_1,LSR_2,LSR_3,} смежных с граничным маршрутизатором LER_1 и граничными с множеством остальных сетевых маршрутизаторов. С этой целью маршрутизатор LER_1 направляет всем смежным маршрутизаторам управляющий пакет «запрос метки» с указанием префикса A(LER_1\LER_4), указывающего на свой адрес и адрес получателя. На основании этой информации маршрутизатор LSR_1 формирует запись в своей табли-

це меток (табл.3.1). В качестве исходящей метки выбирается первая свободная метка, например, метка 1.

Таблица 3.1

Таблица меток маршрутизатора LSR_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1\LER_4)	1	1

Маршрутизатор LSR_2 формирует запись в своей таблице меток (табл.3.2). В качестве исходящей метки выбирается первая свободная метка, например, метка 4.

Таблица 3.2

Таблица меток маршрутизатора LSR_2				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	4	2

Маршрутизатор LSR_3 формирует запись в своей таблице меток (табл.3.3). В качестве исходящей метки выбирается первая свободная метка, например, метка 2.

Таблица 3.3

Таблица меток маршрутизатора LSR_3				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	2	1

В свою очередь маршрутизатор LER_1, формирует запись в своей таблице меток (табл.3.4) на основании выходных меток из пакетов, полученных с маршрутизаторов LSR_1, LSR_2, LSR_3.

Таблица 3.4

Таблица меток маршрутизатора LER_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
1	1	A(LER_1)	-----	-------
4	2	A(LER_1)	----	------

2	1	A(LER_1)	-----	-------

Таким образом, формируется подсеть SN_1 с множеством вершин $\{LSR_0,LSR_1,LSR_2,LSR_3\}$ и тремя маршрутами передачи: $M_1=(LSR_1 \rightarrow LER_1)$; $M_2=(LSR_2 \rightarrow LER_1)$; $M_3=(LSR_3 \rightarrow LER_1)$ (рис. 3.5).

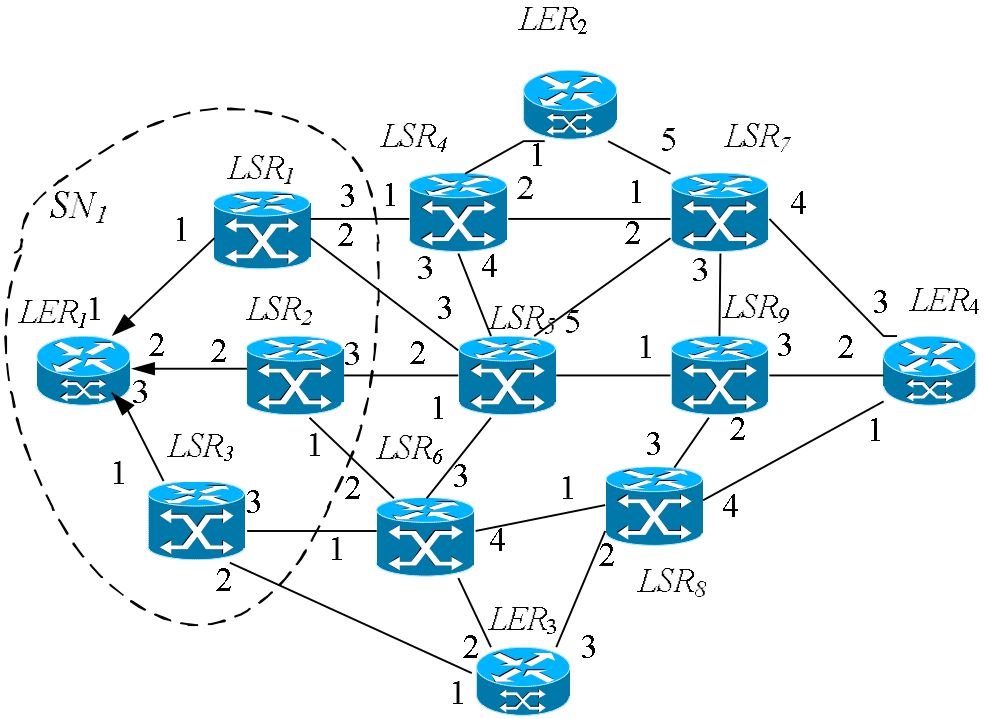


Рис. 3.5. Первый этап формирования подсети SN_1

Затем среди множества граничных маршрутизаторов $BR=\{LSR_1,LSR_2,LSR_3\}$ выбирается маршрутизатор с минимальным числом внешних каналов относительно сетевых маршрутизаторов подсети $SN_2=SN_0 \backslash SN_1$. В данном случае это сетевой маршрутизатор LSR_3, который связан только с одним внешним сетевым маршрутизатором LSR_6.

Сетевой маршрутизатор LSR_3 посылает маршрутизатор LSR_6 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LSR_6 формирует запись в своей таблице меток (табл.3.5). В качестве исходящей метки выбирается первая свободная метка, например, метка 5.

Таблица 3.5

Таблица меток маршрутизатора LSR_6				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт

-----	---	A(LER_1)	5	1

Сетевой маршрутизатор LSR_3 становится внутренним маршрутизатором, а маршрутизатор LSR_6 становится граничным маршрутизатором для подсети SN_1 (рис. 3.6).

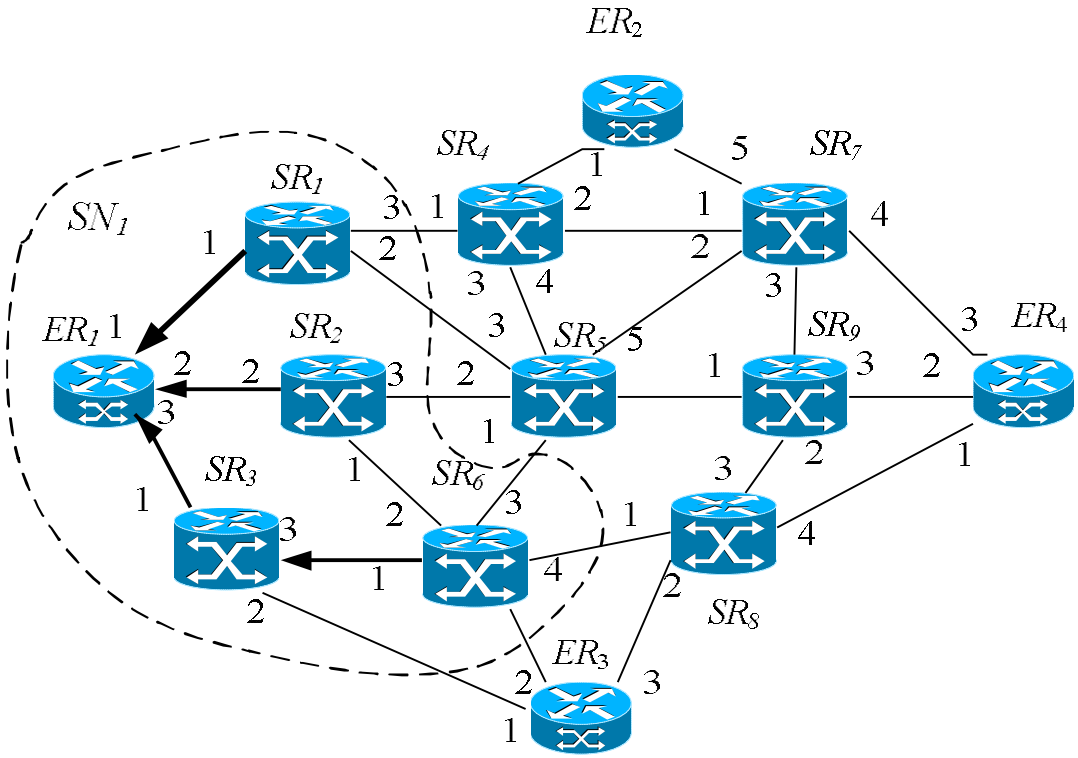


Рис. 3.6 Второй этап формирования подсети SN_1

Сетевой маршрутизатор LSR_6 направляет маршрутизатору LSR_3 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_3 (табл. 3.6) и формируется путь M_3=(LSR_6→LSR_3→LER_1).

Таблица 3.6

Таблица меток маршрутизатора LSR_3				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
5	3	A(LER_1)	2	1

Таким образом, формируется подсеть SN_1 с множеством вершин {LSR_0,LSR_1,LSR_2,LSR_3,LSR_6} и тремя маршрутами передачи: M_1=(LSR_1→LER_1); M_2=(LSR_2→LER_1); M_3=(LSR_6→LSR_3→LER_1).

Затем среди множества граничных маршрутизаторов BR={LSR_1,LSR_2,LSR_6} выбирается маршрутизатор с минимальным числом

внешних каналов относительно сетевых маршрутизаторов подсети $SN_2=SN_0\backslash SN_1$. В данном случае это сетевой маршрутизатор LSR_2, который связан только с одним внешним сетевым маршрутизатором LSR_5.

Сетевой маршрутизатор LSR_2 посылает маршрутизатор LSR_5 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LSR_5 формирует запись в своей таблице меток (табл. 3.7). В качестве выходящей метки выбирается первая свободная метка, например, метка 1.

Таблица 3.7

Таблица меток маршрутизатора LSR_5				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	1	2

Сетевой маршрутизатор LSR_2 становится внутренним маршрутизатором, а маршрутизатор LSR_5 становится граничным маршрутизатором для подсети SN_1 (рис. 3.7).

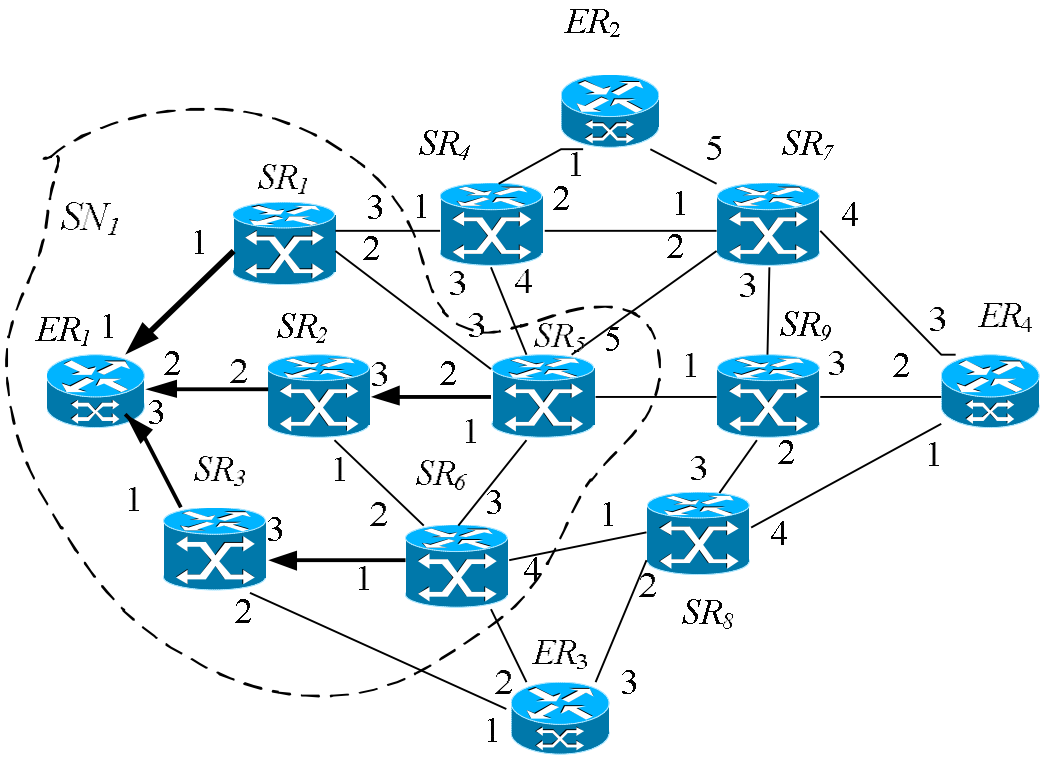


Рис. 3.7. Третий этап формирования подсети SN_1

Сетевой маршрутизатор LSR_5 направляет маршрутизатору LSR_2 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_2 (табл. 3.8) и формируется путь M_3=($LSR_5 \rightarrow LSR_2 \rightarrow LER_1$).

Таблица 3.8

Таблица меток маршрутизатора LSR_2				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
1	3	A(LER_1)	4	2

Таким образом, формируется подсеть SN_1 с множеством вершин $\{LSR_0, LSR_1, LSR_2, LSR_3, LSR_5, LSR_6\}$ и тремя маршрутами передачи: M_1=($LSR_1 \rightarrow LER_1$); M_2=($LSR_5 \rightarrow LSR_2 \rightarrow LER_1$); M_3=($LSR_6 \rightarrow LSR_3 \rightarrow LER_1$).

Затем среди множества граничных маршрутизаторов $BR = \{LSR_1, LSR_5, LSR_6\}$ выбирается маршрутизатор с минимальным числом внешних каналов относительно сетевых маршрутизаторов подсети $SN_2=SN_0 \backslash SN_1$. В данном случае это сетевой маршрутизатор LSR_1, который связан только с одним внешним сетевым маршрутизатором LSR_4.

Сетевой маршрутизатор LSR_1 посылает маршрутизатору LSR_4 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LSR_4 формирует запись в своей таблице меток (табл. 3.9). В качестве выходящей метки выбирается первая свободная метка, например, метка 6.

Таблица 3.9

Таблица меток маршрутизатора LSR_4				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	6	1

Сетевой маршрутизатор LSR_1становится внутренним маршрутизатором, а маршрутизатор LSR_4 становится граничным маршрутизатором для подсети SN_1 (рис. 3.8).

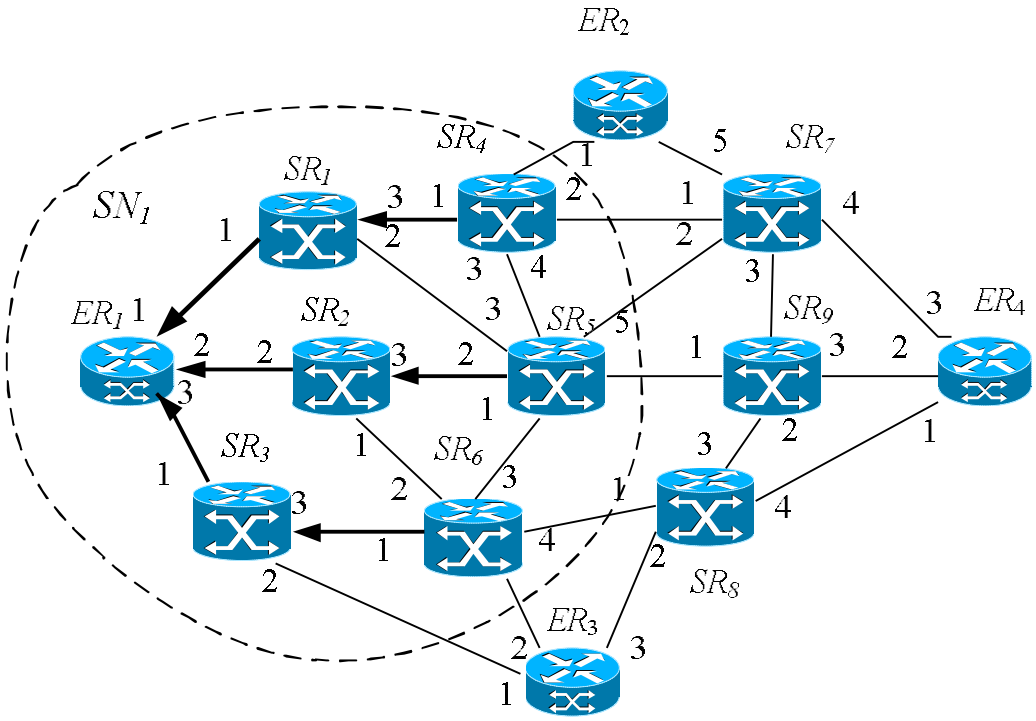


Рис. 3.8 Четвертый этап формирования подсети SN_1

Сетевой маршрутизатор LSR_4 направляет маршрутизатору LSR_1 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_1 (табл. 10) и формируется путь M_1=(LSR_4→LSR_1→LER_1).

Таблица 3.10

Таблица меток маршрутизатора LSR_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
6	3	A(LER_1)	1	1

Таким образом, формируется подсеть SN_1 с множеством вершин {LSR_0, LSR_1,LSR_2,LSR_3,LSR_4,LSR_6} и тремя маршрутами передачи: M_1=(LSR_4→LSR_1→LER_1); M_2=(LSR_5→LSR_2→LER_1); M_3=(LSR_6→LSR_3→LER_1).

Затем среди множества граничных маршрутизаторов BR={LSR_4,LSR_5,LSR_6} выбирается маршрутизатор с минимальным числом внешних каналов относительно сетевых маршрутизаторов подсети SN_2=SN_0\SN_1. В данном случае это сетевой маршрутизатор LSR_6, который связан только с одним внешним сетевым маршрутизатором LSR_8.

Сетевой маршрутизатор LSR_6 посылает маршрутизатор LSR_8 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LSR_8 формирует запись в своей таблице меток (табл. 3.11). В качестве выходящей метки выбирается первая свободная метка, например, метка 10.

Таблица 3.11

Таблица меток маршрутизатора LSR_8				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	10	1

Сетевой маршрутизатор LSR_6 становится внутренним маршрутизатором, а маршрутизатор LSR_8 становится граничным маршрутизатором для подсети SN_1 (рис. 3.9).

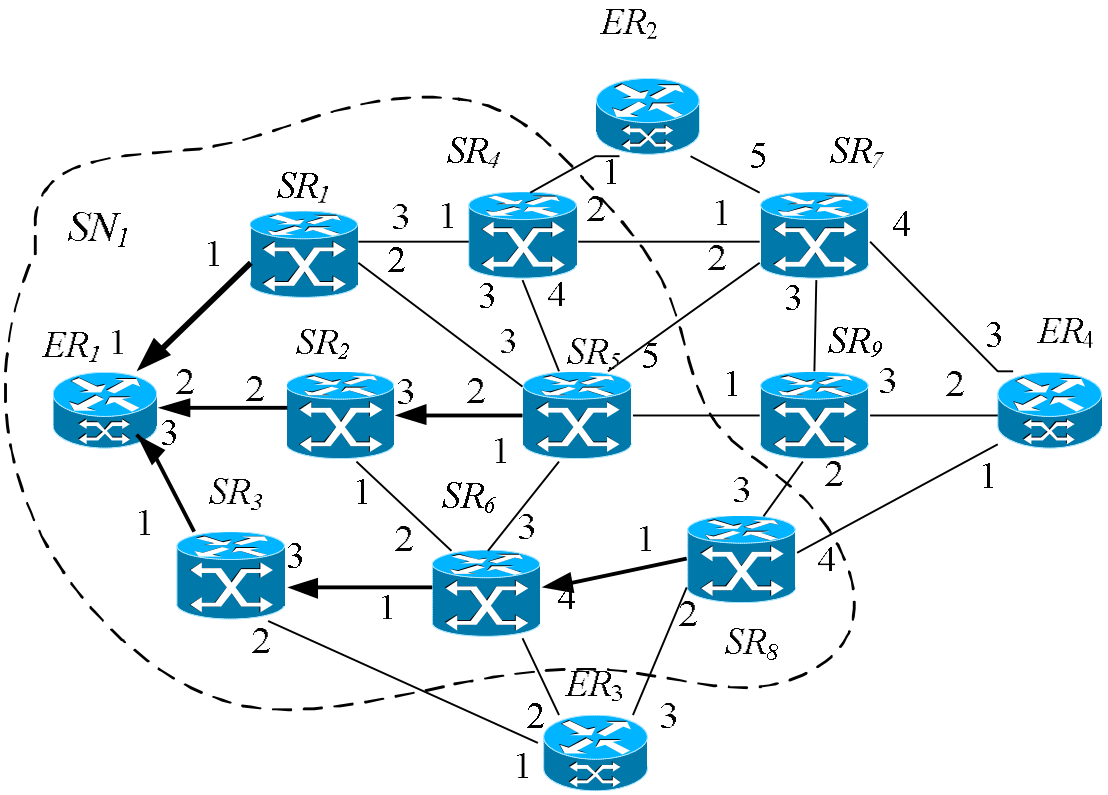


Рис. 3.9 Пятый этап формирования подсети SN_1

Сетевой маршрутизатор LSR_8 направляет маршрутизатору LSR_6 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_6 (табл. 3.12) и формируется путь M_3=(LSR_8→LSR_6→LSR_3→LER_1).

Таблица 3.12

Таблица меток маршрутизатора LSR_6				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
10	4	A(LER_1)	5	1

Таким образом, формируется подсеть SN_1 с множеством вершин $\{LSR_0,LSR_1,LSR_2,LSR_3,LSR_4,LSR_5,LSR_6,LSR_8\}$ и тремя маршрутами передачи:

$M_1=(LSR_4 \rightarrow LSR_1 \rightarrow LER_1)$;

$M_2=(LSR_5 \rightarrow LSR_2 \rightarrow LER_1)$;

$M_3=(LSR_8 \rightarrow LSR_6 \rightarrow LSR_3 \rightarrow LER_1)$.

Среди множества граничных маршрутизаторов $BR=\{LSR_4,LSR_5,LSR_8\}$ находится маршрутизатор LSR_8, непосредственно связанный с маршрутизатором LER_4. Сетевой маршрутизатор LSR_8 посылает маршрутизатору LER_4 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LER_4 формирует запись в своей таблице меток (табл. 13). В качестве выходящей метки выбирается первая свободная метка, например, метка 5.

Таблица 13

Таблица меток маршрутизатора LER_4				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	5	1

Сетевой маршрутизатор LSR_8 становится внутренним маршрутизатором для подсети SN_1 (рис. 3.10).

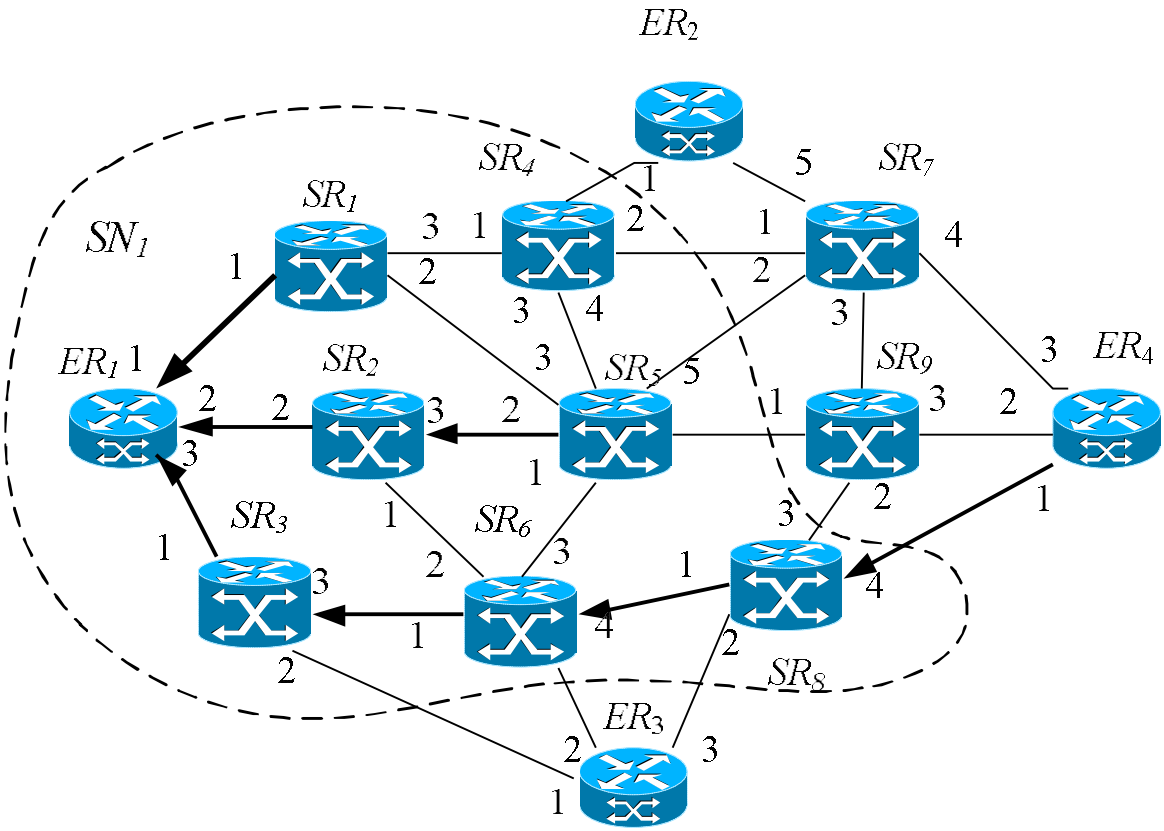


Рис. 3.10 Шестой этап формирования подсети SN_1

Маршрутизатор LER_4 направляет маршрутизатору LSR_8 пакет с выходящей меткой и префиксом А(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_8 (табл. 3.14) и формируется путь M_3=(LER_4→LSR_8→LSR_6→LSR_3→LER_1).

Таблица 3.14

Таблица меток маршрутизатора LSR_8				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
5	4	А(LER_1)	10	1

Для формирования пути в обратном направлении M_3=(LER_1→LSR_3→LSR_6→LSR_8→LER_4) маршрутизатор LER_4 передает по пути M_3=(LER_4→LSR_8→LSR_6→LSR_3→LER_1) установочный пакет со своим префиксом А(LER_4).

Таблица 3.15

Таблица меток маршрутизатора LER_4				
Входящая	Входящий	Префикс	Выходящая	Выходящий

метка	порт		метка	порт
-----	---	A(LER_1)	5	1
7	1	A(LER_4)		

В результате в таблицах меток промежуточных узлов пути M_3=($LER_4 \rightarrow LSR_8 \rightarrow LSR_6 \rightarrow LSR_3 \rightarrow LER_1$) формируются соответствующие записи:

Таблица 3.16

Таблица меток маршрутизатора LSR_8				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
5	4	A(LER_1)	10	1
2	1	A(LER_4)	7	4

Таблица 3.17

Таблица меток маршрутизатора LSR_6				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
10	4	A(LER_1)	5	1
3	1	A(LER_4)	2	4

Таблица 3.18

Таблица меток маршрутизатора LSR_3				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
5	3	A(LER_1)	2	1
4	1	A(LER_4)	3	3

Таблица 3.19

Таблица меток маршрутизатора LER_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
1	1	A(LER_1)	-----	-------
4	2	A(LER_1)	----	------
2	1	A(LER_1)	-----	-------
---	-----	A(LER_4)	4	3

Затем среди оставшихся граничных маршрутизаторов BR={LSR_4,LSR_5} выбирается маршрутизатор с минимальным числом внешних каналов относительно сетевых маршрутизаторов подсети $SN_2=SN_0 \backslash SN_1$. В данном случае

это сетевой маршрутизатор LSR_4, который связан только с одним внешним сетевым маршрутизатором LSR_7.

Сетевой маршрутизатор LSR_4 посылает маршрутизатору LSR_7 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LSR_7 формирует запись в своей таблице меток (табл. 3.20). В качестве выходящей метки выбирается первая свободная метка, например, метка 10.

Таблица 3.20

Таблица меток маршрутизатора LSR_7				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	10	1

Сетевой маршрутизатор LSR_4 становится внутренним маршрутизатором, а маршрутизатор LSR_7 становится граничным маршрутизатором для подсети SN_1 (рис. 3.11).

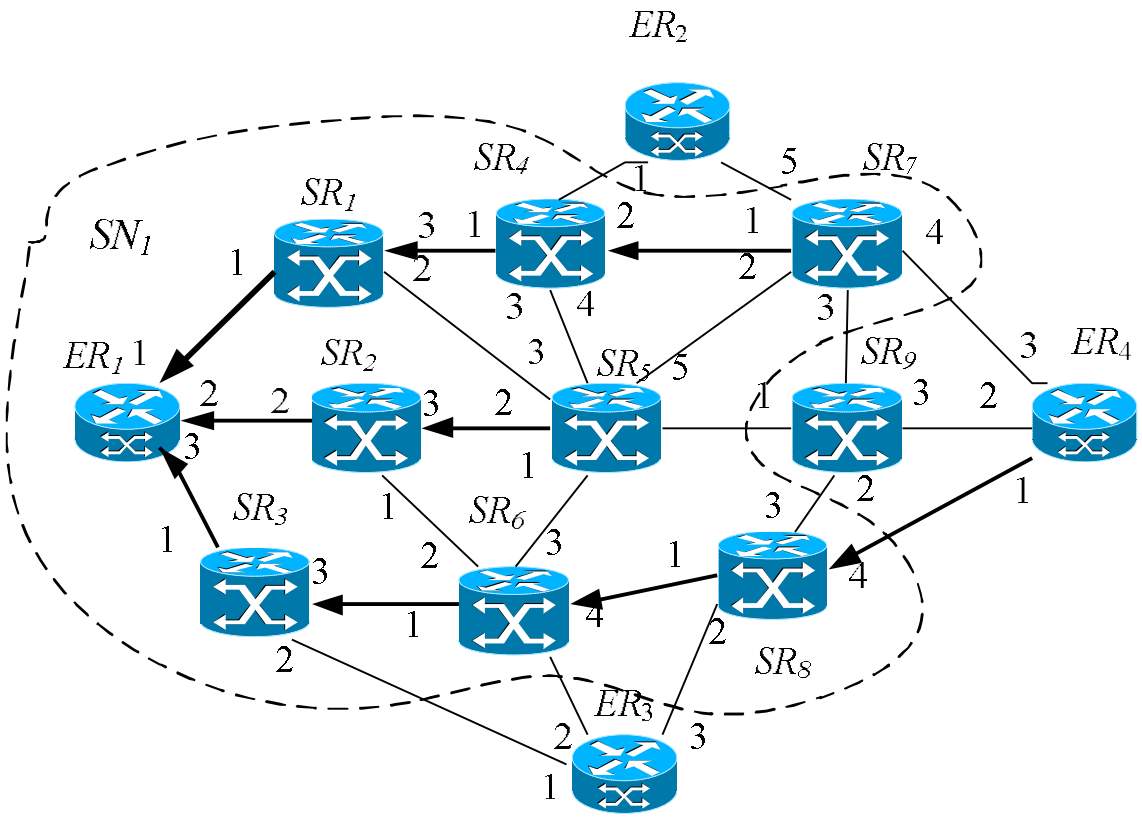


Рис. 3.11 Седьмой этап формирования подсети SN_1

Сетевой маршрутизатор LSR_7 направляет маршрутизатору LSR_4 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации

корректируется таблица меток маршрутизатора LSR_4 (табл. 16) и формируется путь M_1=(LSR_7→LSR_4→LSR_1→LER_1).

Таблица 3.21

Таблица меток маршрутизатора LSR_4				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
10	2	A(LER_1)	5	2

Таким образом, формируется подсеть SN_1 с множеством вершин {LSR_0,LSR_1,LSR_2,LSR_3,LSR_4,LSR_5,LSR_6,LSR_7,LSR_8} и тремя маршрутами передачи:

M_1=(LSR_7→LSR_4→LSR_1→LER_1);

M_2=(LSR_5 →LSR_2 → LER_1);

M_3=(LER_4→LSR_8→LSR_6→LSR_3→LER_1).

Среди множества граничных маршрутизаторов BR={LSR_5,LSR_7} находится маршрутизатор LSR_7, непосредственно связанный с маршрутизатором LER_4. Сетевой маршрутизатор LSR_7 посылает маршрутизатору LER_4 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LER_4 формирует запись в своей таблице меток (табл. 3.22). В качестве выходящей метки выбирается первая свободная метка, например, метка 11.

Таблица 3.22

Таблица меток маршрутизатора LER_4				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	11	3

Сетевой маршрутизатор LSR_7 становится внутренним маршрутизатором для подсети.

Маршрутизатор LER_4 направляет маршрутизатору LSR_7 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_7 (табл. 3.23) и формируется путь M_1=(LER_4→LSR_7→LSR_4→LSR_1→LER_1).

Таблица 3.23

Таблица меток маршрутизатора LSR_7				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
11	4	A(LER_1)	10	1

Для формирования пути в обратном направлении $M0_1$=($LER_1 \rightarrow LSR_1 \rightarrow LSR_4 \rightarrow LSR_7 \rightarrow LER_4$) маршрутизатор LER_4 передает по пути M_1=($LER_4 \rightarrow LSR_7 \rightarrow LSR_4 \rightarrow LSR_4 \rightarrow LER_1$) установочный пакет со своим префиксом A(LER_4). В результате в таблицах меток промежуточных узлов {LSR_7,LSR_4,LSR_1} формируются соответствующие записи:

Таблица 3.24

Таблица меток маршрутизатора LSR_7				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
11	4	A(LER_1)	10	1
5	1	A(LER_1)	10	4

Таблица 3.25

Таблица меток маршрутизатора LSR_4				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
10	2	A(LER_1)	5	1
10	1	A(LER_4)	5	2

Таблица 3.26

Таблица меток маршрутизатора LSR_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
6	3	A(LER_1)	1	1
5	1	A(LER_4)	10	3

Таблица 3.27

Таблица меток маршрутизатора LER_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
1	1	A(LER_1)	-----	-------

4	2	A(LER_1)	----	------
2	1	A(LER_1)	-----	-------
---	-----	A(LER_4)	4	3
		A(LER_4)	5	1

В результате в подсети SN_1 остается только один граничный маршрутизатор LSR_5, который связан только с одним внешним сетевым маршрутизатором LSR_9. Сетевой маршрутизатор LSR_5 посылает маршрутизатору LSR_9 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LSR_9 формирует запись в своей таблице меток (табл. 3.28). В качестве выходящей метки выбирается первая свободная метка, например, метка 21.

Таблица 3.28

Таблица меток маршрутизатора LSR_9				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	21	1

Сетевой маршрутизатор LSR_5 становится внутренним маршрутизатором, а маршрутизатор LSR_9 становится граничным маршрутизатором для подсети SN_1 (рис. 3.12).

Рис.3.12

Сетевой маршрутизатор LSR_9 направляет маршрутизатору LSR_5 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_5 (табл. 2.9) и формируется путь M_2=($LSR_9 \rightarrow LSR_5 \rightarrow LSR_2 \rightarrow LER_1$).

Таблица 3.29

Таблица меток маршрутизатора LSR_5				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
21	5	A(LER_1)	1	2

Таким образом, формируется подсеть SN_1 с множеством вершин {LSR_0,LSR_1,LSR_2,LSR_3,LSR_4,LSR_5,LSR_6,LSR_7,LSR_8,LSR_9} и тремя маршрутами передачи:

M_1=(LER_4→LSR_7→LSR_4→LSR_1→LER_1);

M_2=(LSR_9→LSR_5→LSR_2→LER_1);

M_3=(LER_4→LSR_8→LSR_6→LSR_3→LER_1).

Маршрутизатор LSR_9, непосредственно связанный с маршрутизатором LER_4. Сетевой маршрутизатор LSR_9 посылает маршрутизатору LER_4 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LER_4 формирует запись в своей таблице меток (табл. 3.30). В качестве выходящей метки выбирается первая свободная метка, например, метка 4.

Таблица 3.30

Таблица меток маршрутизатора LER_4				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	11	2

Маршрутизатор LER_4 направляет маршрутизатору LSR_9 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_9 (табл. 3.32) и формируется путь M_2=(LER_4→LSR_9→LSR_5→LSR_2→LER_1).

Таблица 3.31

Таблица меток маршрутизатора LSR_9				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
11	3	A(LER_1)	21	1

Для формирования пути в обратном направлении M0$_2$=(LER_1→LSR_2→LSR_5→LSR_9→LER_4) маршрутизатор LER_4 передает по пути M_2=(LER_4→LSR_9→LSR_5→LSR_2→LER_1) установочный пакет со своим префиксом A(LER_4). В результате в таблицах меток промежуточных узлов {LSR_9,LSR_5,LSR_2} формируются соответствующие записи:

Таблица 3.32

Таблица меток маршрутизатора LSR_9				
Входящая	Входящий	Префикс	Выходящая	Выходящий

метка	порт		метка	порт
11	3	A(LER_1)	21	1
1	1	A(LER_4)	21	3

Таблица 3.33

Таблица меток маршрутизатора LSR_5				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
21	5	A(LER_1)	1	2
4	2	A(LER_4)	1	5

Таблица 3. 34

Таблица меток маршрутизатора LSR_2				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
1	3	A(LER_1)	4	2
8	2	A(LER_4)	4	3

Таблица 3.35

Таблица меток маршрутизатора LER_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
1	1	A(LER_1)	-----	-------
4	2	A(LER_1)	----	------
2	3	A(LER_1)	-----	-------
---	-----	A(LER_4)	4	3
-----	-----	A(LER_4)	5	1
		A(LER_4)	8	2

Таблица 3.36

Таблица меток маршрутизатора LER_4				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
----- LSR8	---	A(LER_1)	5	1
7	1	A(LER_4)	--	--
--	---	A(LER_1)	11	3
21	3	A(LER_4)	---	--
--	--	A(LER_1)	11	2

5	2	A(*LER4*)	--	--

Таким образом, в маршрутизаторе *LER1* формируется три двунаправленные пути между маршрутизаторами *LER1* и *LER4*:

- первый путь *M1* с параметрами (порт=1; входящая метка =1; выходящая метка =5);
- второй путь *M2* с параметрами (порт=2; входящая метка =4; выходящая метка =8);
- третий путь *M3* с параметрами (порт=3; входящая метка =2; выходящая метка =4);

Количество путей зависит от степени вершин отправителя и получателя. При степени вершины *LER4* равной двум, например, отсутствует связь между маршрутизаторами *LER4* и *LSR9*. В этом случае путь M_2=(*LSR9*→*LSR5*→*LSR2*→*LER1*).

Таблица меток маршрутизатора *LSR9* будет иметь следующий вид (табл. 3.37), в ней отсутствует указание входного порта и входной метки со стороны маршрутизатора *LER4*.

Таблица 3.37

Таблица меток маршрутизатора *LSR9*				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
---	----	A(*LER1*)	21	1

В этом случае маршрутизатор *LER4* не может сформировать путь *M2*=(*LER4*→*LSR9*→*LSR5*→*LSR2*→*LER1*) и передать по нему установочный пакет со своим префиксом A(*LER4*). Соответственно в таблице меток маршрутизатора *LER1* удаляется строка, формирующая путь M_2=(*LER1*→*LSR2*→*LSR5*→*LSR9*→*LER4*) (табл. 3.38).

Таблица 3.38

Таблица меток маршрутизатора *LER1*				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
1	1	A(*LER1*)	-----	-------

~~4~~	~~2~~	~~A(LER_1)~~	~~---~~	~~-----~~
2	3	A(LER_1)	-----	-------
---	-----	A(LER_4)	4	3
-----	-----	A(LER_4)	5	1

Временная сложность данного алгоритма равна O(S_mD), где S_m – средняя степень вершин и D – диаметр графа.

Данный алгоритм позволяет одновременно формировать множество путей от одного граничного маршрутизатора к нескольким граничным маршрутизаторам. В этом случае формируется стек меток, содержащий префиксы соответствующих маршрутов.

В качестве примера рассмотрим формирование каналов между маршрутизатором *LER* и маршрутизаторами LER_2, LER_3, LER_4.

С этой целью маршрутизатор LER_1 направляет всем смежным маршрутизаторам {LSR_1,LSR_2,LSR_3} управляющие пакеты «запрос метки» с указанием префиксов {A(LER_1\$LER_2$)|$N$}, {A($LER_1$\$LER_3$)|$N$}, {A($LER_1$\$LER_4$)|$N$}, где: N=1,2,3 – номер подканала. В данном случае номер подканала будет совпадать с номером входящих портов маршрутизатора LER_1.

На основании этой информации маршрутизатор LSR_1 формирует запись в своей таблице меток (табл. 3.39). В качестве исходящей метки выбирается первая свободная метка, например, метка 1.

Таблица 3.39

Таблица меток маршрутизатора LSR_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1\LER_2)|1	1	1
		A(LER_1\LER_3)|1	2	1
		A(LER_1\LER_4)|1	3	1

Маршрутизатор LSR_2 формирует запись в своей таблице меток (табл. 3.40). В качестве исходящей метки выбирается первая свободная метка, например, метка 4.

Таблица 3.40

Таблица меток маршрутизатора LSR_2				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1\LER_2)|2	4	2
--	--	A(LER_1\LER_3)|2	5	2
--	--	A(LER_1\LER_4)|2	6	2

Маршрутизатор LSR_3 формирует запись в своей таблице меток (табл. 3.41). В качестве исходящей метки выбирается первая свободная метка, например, метка 2.

Таблица 3.41

Таблица меток маршрутизатора LSR_3				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1\LER_2)|3	7	3
---	---	A(LER_1\LER_3)|3	8	3
---	---	A(LER_1\LER_4)|3	9	3

В свою очередь маршрутизатор LER_1, формирует запись в своей таблице меток (табл. 3.42) на основании выходных меток из пакетов, полученных с маршрутизаторов LSR_1,LSR_2,LSR_3.

Таблица 3.42

Таблица меток маршрутизатора LER_1				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
1	1	A(LER_1\LER_2)|1	-----	-------
2	1	A(LER_1\LER_3)|1	----	------
3	1	A(LER_1\LER_4)|1	-----	-------
4	2	A(LER_1\LER_2)|2	--	--
5	2	A(LER_1\LER_3)|2	--	--
6	2	A(LER_1\LER_4)|2	--	--
7	3	A(LER_1\LER_2)|3	--	--
8	3	A(LER_1\LER_3)|3	--	--
9	3	A(LER_1\LER_4)|3	--	--

Таким образом, формируется подсеть SN_1 с множеством вершин $\{LSR_0,LSR_1,LSR_2,LSR_3\}$ и тремя маршрутами передачи: $M_1=(LSR_1 \rightarrow LER_1)$; $M_2=(LSR_2 \rightarrow LER_1)$; $M_3=(LSR_3 \rightarrow LER_1)$ (рис. 3.42).

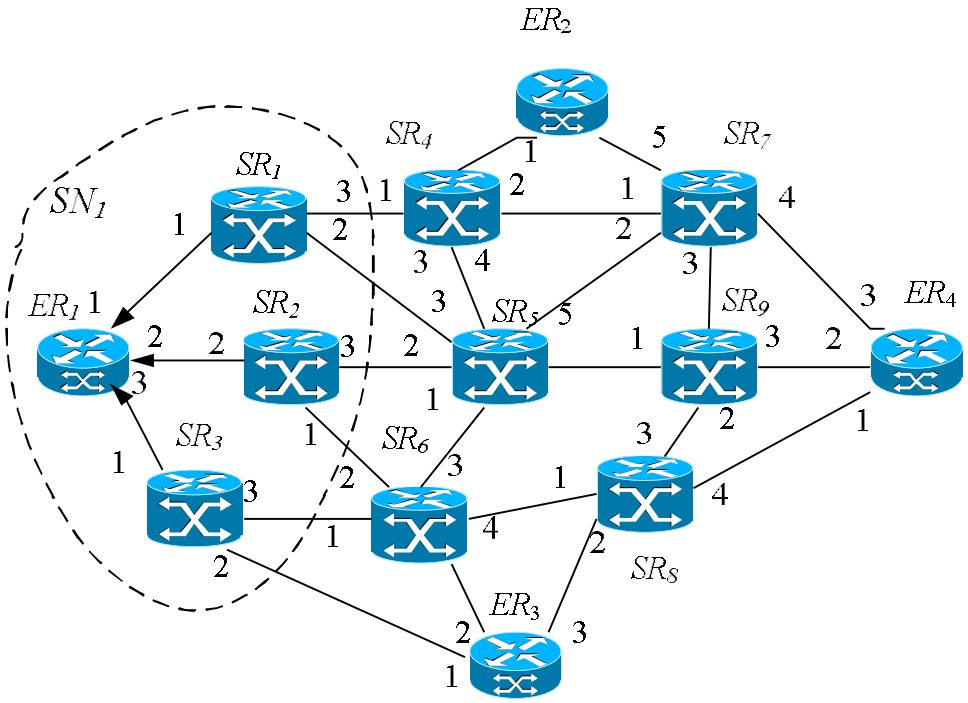


Рис. 3.12. Восьмой этап формирования подсети SN_1

Затем среди множества граничных маршрутизаторов $BR=\{LSR_1,LSR_2,LSR_3\}$ выбирается маршрутизатор с минимальным числом внешних каналов относительно сетевых маршрутизаторов подсети $SN_2=SN_0 \backslash SN_1$. В данном случае это сетевой маршрутизатор LSR_3, который связан только с одним внешним сетевым маршрутизатором LSR_6.

Сетевой маршрутизатор LSR_3 посылает маршрутизатор LSR_6 управляющий пакет «запрос метки» с указанием префикса A(LER_1) граничного маршрутизатора LER_1. На основании этой информации маршрутизатор LSR_6 формирует запись в своей таблице меток (табл. 3.43). В качестве исходящей метки выбирается первая свободная метка, например, метка 5.

Таблица 3.43

Таблица меток маршрутизатора LSR_6				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
-----	---	A(LER_1)	5	1

Сетевой маршрутизатор LSR_3 становится внутренним маршрутизатором, а маршрутизатор LSR_6 становится граничным маршрутизатором для подсети SN_1 (рис. 3.14).

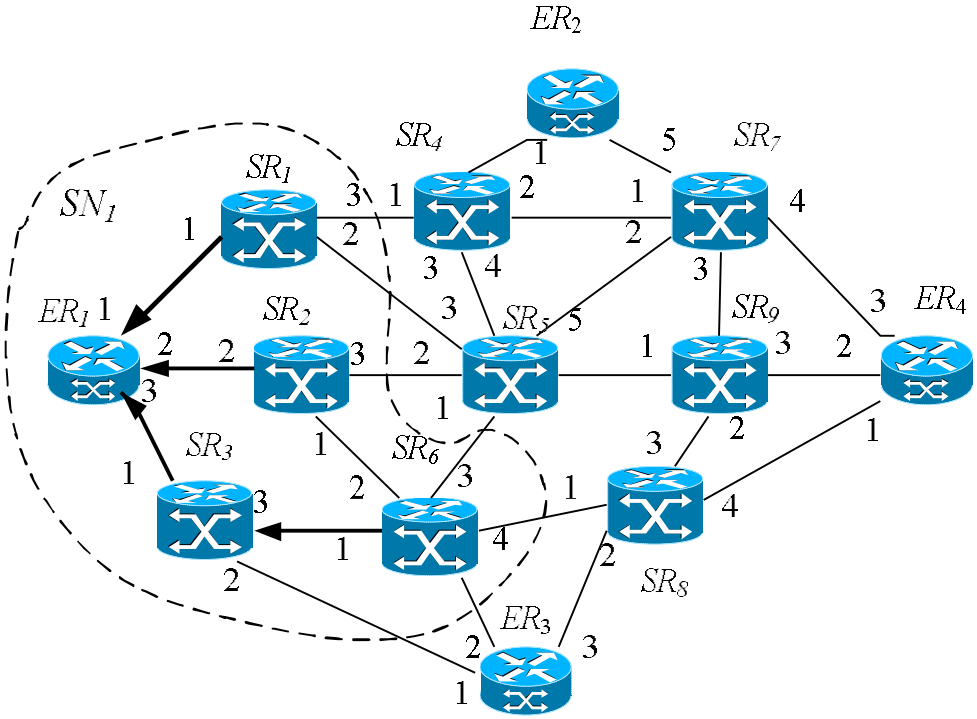


Рис. 3.13 Завершающий этап формирования подсети SN_1

Сетевой маршрутизатор LSR_6 направляет маршрутизатору LSR_3 пакет с выходящей меткой и префиксом A(LER_1). На основании этой информации корректируется таблица меток маршрутизатора LSR_3 (табл. 3.44) и формируется путь M_3=(LSR_6→LSR_3→LER_1).

Таблица 3.44

Таблица меток маршрутизатора LSR_3				
Входящая метка	Входящий порт	Префикс	Выходящая метка	Выходящий порт
5	3	A(LER_1)	2	1

Таким образом, формируется подсеть SN_1 с множеством вершин {LSR_0,LSR_1,LSR_2,LSR_3,LSR_6} и тремя маршрутами передачи: M_1=(LSR_1→LER_1); M_2=(LSR_2→LER_1); M_3=(LSR_6→LSR_3→LER_1).

ВЫВОДЫ ПО ГЛАВЕ 3

1. Предложен способ формирования таблиц меток для организации многопутевой маршрутизации в сетях MPLS. Данный способ позволяет одновременно формировать все множество непересекающихся путей между двумя граничными маршрутизаторами LER_i и LER_j.

2. В данном случае временная сложность формирования таблиц меток маршрутов существенно не зависит от количества множества сформированных каналов передачи и определяется величиной $O(N+M+L)$, где: N – число маршрутизаторов LSR_i, M – число каналов связи, L – общая длина всех сформированных путей между маршрутизаторами LER_i и LER_j. Это существенно меньше, чем временная сложность $O(qN^3)$ последовательного формирования q путей с помощью алгоритма Дейкстры.

3. Данный способ также позволяет сформировать множество непересекающихся каналов от одного граничного маршрутизатора к заданному множеству других граничных маршрутизаторов, образовав виртуальную сеть поверх сети MPLS.

ГЛАВА 4
ФОРМИРОВАНИЕ МНОГОПУТЕВЫХ КАНАЛОВ В СИСТЕМАХ РАСПРЕДЕЛЕННОЙ ОБРАБОТКИ ИНФОРМАЦИИ

4.1. Способ формирования множества путей в сетевых центрах данных

В настоящее время в связи с возрастающими потребностями в вычислительной мощности и объемов информации актуальными становятся сетевые центры данных (DCN). Функциональным назначением DCN является высокоскоростная обработка и хранение большого объема данных. Сетевые центры данных (DCN) должны обеспечить высокую пропускную способность и надежность передачи информации. Это, в свою очередь, предъявляет высокие требования к конструированию трафика (TE) в таких системах.

При построении современных DCN в основном используется топология Fat Tree [92], которая является одной из наиболее распространенных топологий для построения распределенных систем, ориентированных на решение высокопроизводительных задач. Топология Fat Tree представляет собой дерево, листьями которого являются вычислительные устройства, а узлами – коммутаторы. При этом у коммутаторов более высоких уровней пропускная способность каналов больше, т.е. связи с другими вершинами более «толстые». Поэтому эта топология и получила название Fat Tree (толстое дерево).

DCN при использовании топологии Fat Tree образуют четырех уровневую структуру [93], на верхнем уровне располагаются основные коммутаторы (Core swiches), уровнем ниже располагаются агрегацийные коммутаторы (Aggregation swiches), затем располагаются граничные коммутаторы (Edle swiches) к которым подключаются вычислительные устройства (Hosts) (рис. 4.1).

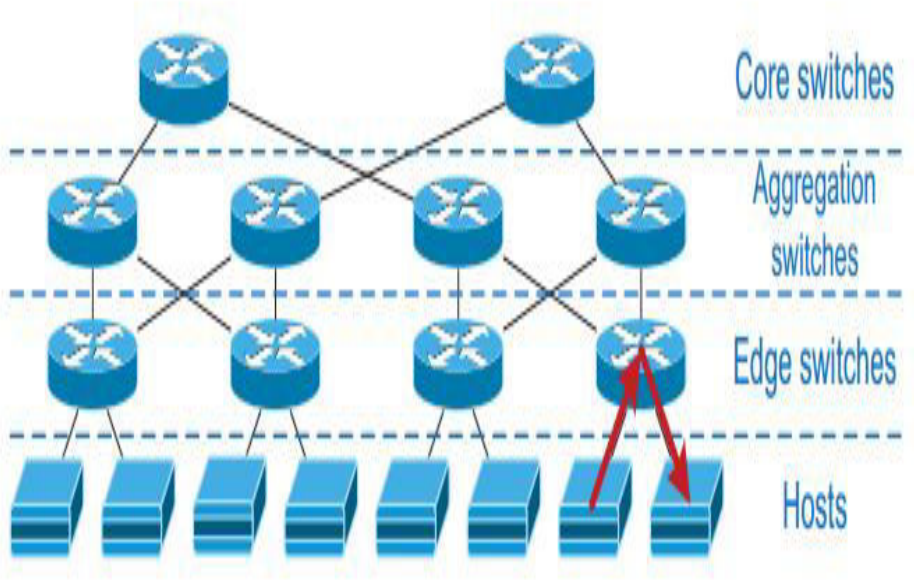


Рис. 4.1. Топология Fat Tree

В свою очередь, для DCN, состоящих из сотен и тысяч серверов большой размерности, топология Fat tree, на основе принципов самоподобия, преобразуется в так называемую сеть Dragonfly [94]. Учет фрактального характера топологии сети Dragonfly позволяет упростить процесс конструирования трафика, в частности процесс маршрутизации. Главными требованиями, предъявляемыми к маршрутизации в DCN, являются надежность, минимальная задержка и отказоустойчивость. Этим критериям в большей степени удовлетворяют способы многопутевой маршрутизации, которые нашли широкое применение в DCN. Одним из основных недостатков комбинаторных алгоритмов многопутевой маршрутизации является значительная их временная сложность, которая, как правило, носит нелинейный характер. В работе [95] предложен волновой алгоритм формирования множества путей между произвольными вершинами с линейной временной сложностью. Дальнейшее уменьшение временной сложности связано с учетом топологии графа сети.

Модифицированный способ доступа на основе многопутевой маршрутизации.

Для топологии Fat tree эффективным является метод поиска в глубину (DFS, сокращение от Depth-first search), в основе которого лежит рекурсивный способ обхода вершин графа. В процессе обхода узлы отмечаются как посещенные и не посещённые, вначале все узлы являются непосвящёнными. Узел, который принимает управляющий пакет отмечается как посещенный и анализирует информацию о смежных с ним узлах. Затем данный узел согласно определенного правила передает пакет одному из неотмеченных смежных с ним узлов. Если у данного узла нет неотмеченных смежных с ним узлов, то он передает управляющий пакет обратно.

К недостаткам базового алгоритма следует отнести то, что он не учитывает:

1) пропускную способность каналов связи;
2) особенности топологии сети.

Существуют три основных стратегии выбора каналов:

- Worst-Fit – выбирается канал з наибольшей доступной пропускной способностью;
- First-Fit – выбирается канал с любой из доступной пропускной способностью, которая отвечает требованиям;
- Best-Fit – выбирается канал с доступной пропускной способностью, которая наилучшим способом отвечает требованиям.

Следует отметить, что при формировании множества путей в DCN с помощью метода DFS наиболее эффективной является стратегия Worst-Fit. Дальнейшее повышение эффективности метода DFS при формировании множества путей связано с учетом особенности иерархической организации DCN и пропускной способности каналов связи между узлами сети.

Метод DFS использует централизованный способ маршрутизации при котором центральный контроллер содержит всю информацию, необходимую для формирования маршрутов. Кроме топологии сети центральный контроллер содержит информацию о пропускной способности каналов связи, которая используется для выбора канала связи на более нижнем уровне или для возврата на уровень выше по дереву. После формирования очередного пути до-

пустимая пропускная способность каналов связи, принадлежащих выбранному пути уменьшается на заданную величину.

На первом шаге алгоритма определяется уровень топологии на котором могут быть объединены узлы отправитель и получатель информации. Это делается на основании того, что центральный контроллер содержит информацию о топологии сети. Если узлы отправитель и получатель информации подключены к одному и тому же граничному коммутатору, то он напрямую связывает эти узлы. В противном случае путь проходит через несколько коммутаторов, количество которых зависит от взаимного расположения узлов.

Определение уровня соединения узлов позволяет упростить процесс маршрутизации. В этом случае исключаются лишние переходы между уровнями.

Для ускорения перехода между уровнями в работе [96] предложен модифицированный алгоритм, в котором вводится глобальная переменная next, определяющая направление обхода узлов сети. При next=1 перемещение должно осуществляется вверх по дереву. При next=-1 следует выбирать путь вниз по дереву. При прохождении через уровень на котором могут быть связаны узлы отправителя и получателя значение переменной next становится равной -1. Это говорит о том, что обход достиг локального максимума и нужно двигаться вниз по дереву.

Если в результате обхода достигается крайний коммутатор, который не связан с получателем, переменная next принимает значение next=1, чтобы вернуться на предыдущий уровень. Использование переменной next является ключевым при реализации алгоритма DFS, так как позволяет учитывать особенности иерархической топологии.

Таким образом, модифицированный алгоритм DFS состоит из следующей последовательности операций:

1. Начало.
2. Определение уровня соединения пути.
3. Next:=1.

4. Установить узел отправитель текущим узлом.
5. Добавить текущий узел к узлам пути.
6. Если текущий узел не является конечным коммутатором, то переход к пункту 9.
7. Если текущий узел является конечным коммутатором получателя, то переход к пункту 14.
8. Установить предыдущий узел текущим.
9. Если достигнут верхний уровень, то установить next = -1.
10. Выбрать следующий узел среди смежных узлов.
11. Если не удалось выбрать следующий узел, то переход к пункту 8.
12. Установить выбранный узел текущим.
13. Переход к пункту 5.
14. Добавить узел получатель к пути.
15. Конец.

4.2. Анализ эффективности модифицированного способа формирования множества путей в сетевых центрах данных

С целью анализа эффективности предлагаемого метода формирования множества путей была разработана специальная система моделирования. На рис.4.2 представлен пример формирования путей между 1 и 5 узлами DCN. Длина *i*-го пути (L_i) определяется количеством переходов между исходной и конечной вершинами и равна N-1, где N – общее число вершин данного пути.

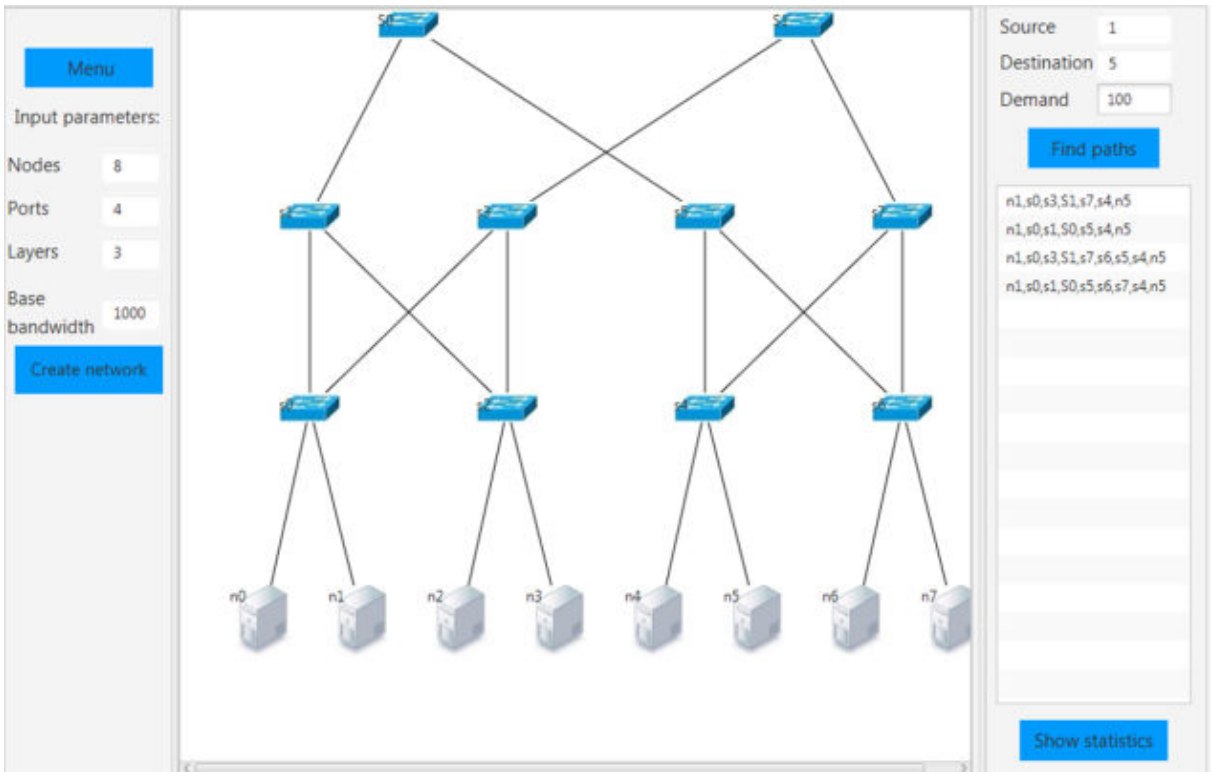


Рис.4.2. Формирование путей в DCN

В результате работы алгоритма формируется четыре пути между вершинами n_1 и n_5:

1. $n_1,s_0,s_3,S1,s_7,s_4,n_5$; $L_1=6$.
2. $n_1,s_0,s_1,S0,s_5,s_4,n_5$; $L_2=6$.
3. $n_1,s_0,s_3,S1,s_7,s_6,s_5,s_4,n_5$; $L_4=8$.
4. $n_1,s_0,s_1,S0,s_5,s_6,s_0,s_4,n_5$; $L_4=8$.

В данном случае средняя длина пути $L_{cp}=7$.

Ниже представлен псевдокод модифицированного алгоритма DFS формирования пути в DCN:

```
DFS(G, a, b, d) {// G: network, a: source, b: destination, d: demand
    1 H=necessary-layer-to-connect(G, a, b);
    2 path={};
    3 u=a; // temp variable indicating current location
    4 next=1; // search direction flag, 1: upstream, -1: downstream
    5 return SEARCH(u, path, next);
    }
    SEARCH(u, path, next) {
```

```
1 path=path+u;
2 if(u=b) return true;
3 if ( layer-of(u)=H) next=−1;// reverse search direction after reaching connecting layer
4 if(next=−1&& layer-of(u)=1) return false;// failure at bottom layer
5 neighbors={v| layer-of(v)= layer-of(u)+next,and available bandwidth of link(u, v)≥d};
6 found=false;
7 while (neighbors!=0&&found=false) {
8 v=worst-fit(neighbors); neighbors=neighbors\{v};
9 found=SEARCH(v,path,next);
10 };
11 return found;}
```

В табл. 4.1 приведена зависимость длины путей от количества узлов DCN, полученная в результате моделирования базового и модифицированного алгоритма маршрутизации.

Таблица 4.1.

Зависимость длины пути от количества узлов DCN

Количество узлов	Длина пути	
	Базовый алгоритм	Модифици-рованный алгоритм
8	9	7
16	15	7
32	27	7
64	51	7
128	99	7
256	195	7
512	387	7
1024	771	7

Из табл. 4.1 видно, что при базовом алгоритме зависимость длины пути от количества узлов носит линейный характер, а для модифицировнного метода остается постоянной. Это связано с тем, что при базовом алгоритме осуществляется последовательный перебор всех путей. При модифицированном алгоритмы учитывается самоподобная топология сети, что позволяет каждый раз выбирать наиболее оптимальный путь и сократить количество итераций.

В табл. 4.2 приведена зависимость числа итераций при обычном и модифицированном алгоритме в зависимости от количества узлов DCN. Как видно из табл. 4.2 преимущество предлагаемого алгоритма увеличивется с ростом количества узлов в сети DCN.

Таблица 4.2.

Зависимость числа итераций от количества узлов DCN

Количество узлов	Число итераций	
	Базовый алгоритм	Модифицированный алгоритм
8	8	6
16	14	10
32	26	18
64	50	34
128	98	66
256	194	130
512	386	258
1024	770	514

4.3. Способ формирования множества MVP в Grid системах

Как правило, в Grid системах используется многоточечное соединение, при котором одна вершина соединяется с несколькими вершинами, образуя звездообразную, древовидную или полносвязную топологию. С целью дальнейшего уменьшения временной сложности формирования виртуальных ка-

налов в данной работе предлагается формировать одновременно виртуальные каналы от одного узла v_i виртуальной сети к множеству смежных узлов $V_i=\{v_j|\forall e_{i,j}\}$ виртуальной сети, представленной в виде графа $G_z(V,E)$. В рамках каждого виртуального канала $e_{i,j}$ между вершинами v_i и v_j формируется множество $W_{i,j}=\{L_r\ |r=1,2,..m\ \}$ непересекающихся физических каналов.

Рассмотрим пример формирования множества путей от вершины от вершины $в_0$ к вершине $в_{14}$.

На рис. 4.3 представлен пример формирования первого пути от вершины $в_0$ к вершине $в_{14}$.

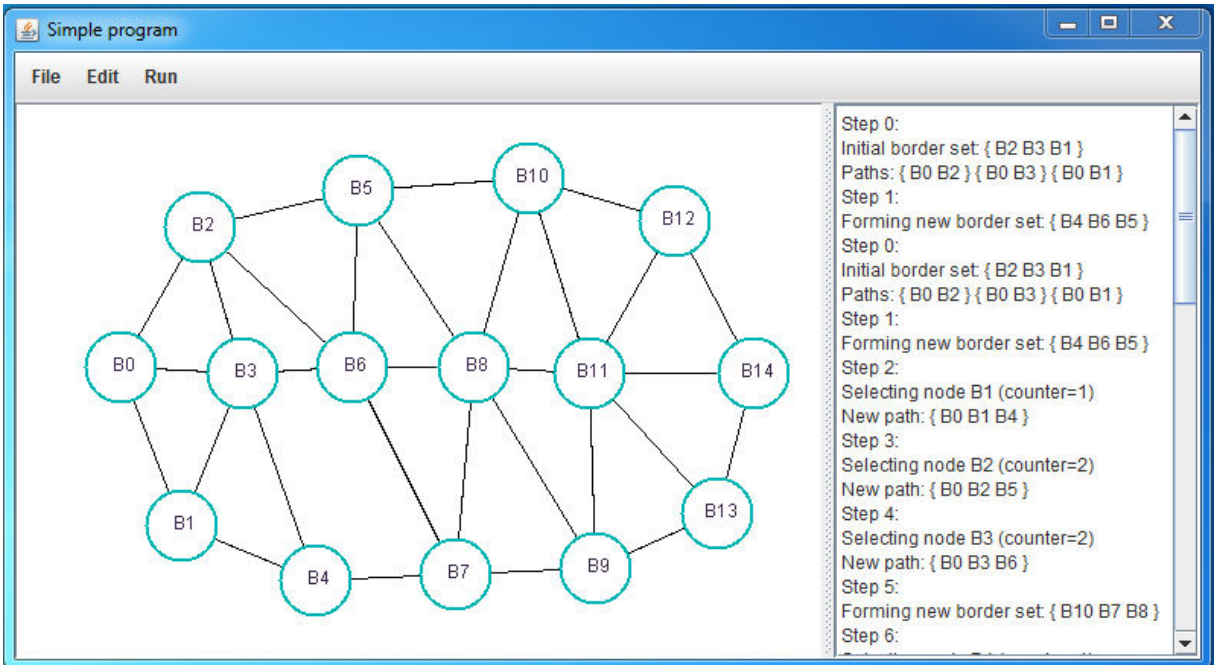


Рис. 4.3. Формирование первого пути от вершины $в_0$ к вершине $в_{14}$.

Формирование всего множества путей между вершинами $в_0$ и $в_{14}$ заключается в следующем:

Step 0:

Initial border set: { B1 B3 B2 } /*формирование множества граничных узлов для вершины $в_0$ */

Paths: { B0 B1 } { B0 B3 } { B0 B2 } /* формирование путей от вершины $в_0$ к вершинам $в_3$,$в_2$,$в_1$. */

Step 1:

Forming new border set: { B4 B6 B5 }/* новое множество граничных узлов */

Step 2:

Selecting node B1 (counter=1) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B1 B4 } /* продолжение пути от вершины $в_0$ к вершине $в_4$ */

Step 3:

Selecting node B3 (counter=2) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B3 B6 } /* продолжение пути от вершины $в_0$ к вершине $в_6$ */

Step 4:

Selecting node B2 (counter=2) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B2 B5 } /* продолжение пути от вершины $в_0$ к вершине $в_5$ */

Step 5:

Forming new border set: { B8 B7 B10 } /* новое множество граничных узлов */

Step 6:

Selecting node B4 (counter=1) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B1 B4 B7 } /* продолжение пути от вершины $в_0$ к вершине $в_7$ */

Step 7:

Selecting node B6 (counter=2) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B3 B6 B8 } /* продолжение пути от вершины $в_0$ к вершине $в_8$ */

Step 8:

Selecting node B5 (counter=2) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B2 B5 B10 } /* продолжение пути от вершины $в_0$ к вершине $в_{10}$ */

Step 9:

Forming new border set: { B11 B9 B12 } /* новое множество граничных узлов */

Step 10:

Selecting node B7 (counter=1) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B1 B4 B7 B9 } /* продолжение пути от вершины $в_0$ к вершине $в_9$ */

Step 11:

Selecting node B8 (counter=2) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B3 B6 B8 B11 } /* продолжение пути от вершины $в_0$ к вершине $в_{11}$ */

Step 12:

Selecting node B10 (counter=2) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B2 B5 B10 B12 } /* продолжение пути от вершины $в_0$ к вершине $в_{12}$ */

Step 13:

Forming new border set: { B13 B14 } /* новое множество граничных узлов */

Step 14:

Selecting node B9 (counter=1) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B1 B4 B7 B9 B13 } /* продолжение пути от вершины $в_0$ к вершине $в_{13}$ */

Step 15:

Selecting node B12 (counter=1) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B2 B5 B10 B12 B14 } /* формирование первого пути от вершины $в_0$ к вершине $в_{14}$ */

Step 16:

Selecting node B11 (counter=2) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B3 B6 B8 B11 B14 } /* формирование второго пути от вершины $в_0$ к вершине $в_{14}$ */

Step 17:

Forming new border set: { B14 } /* новое множество граничных узлов */

Step 18:

Selecting node B13 (counter=1) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B1 B4 B7 B9 B13 B14 }/* формирование третьего пути к вершине $в_{14}$ */

Done

В результате формируются следующая структура графа путей (рис. 4.4).

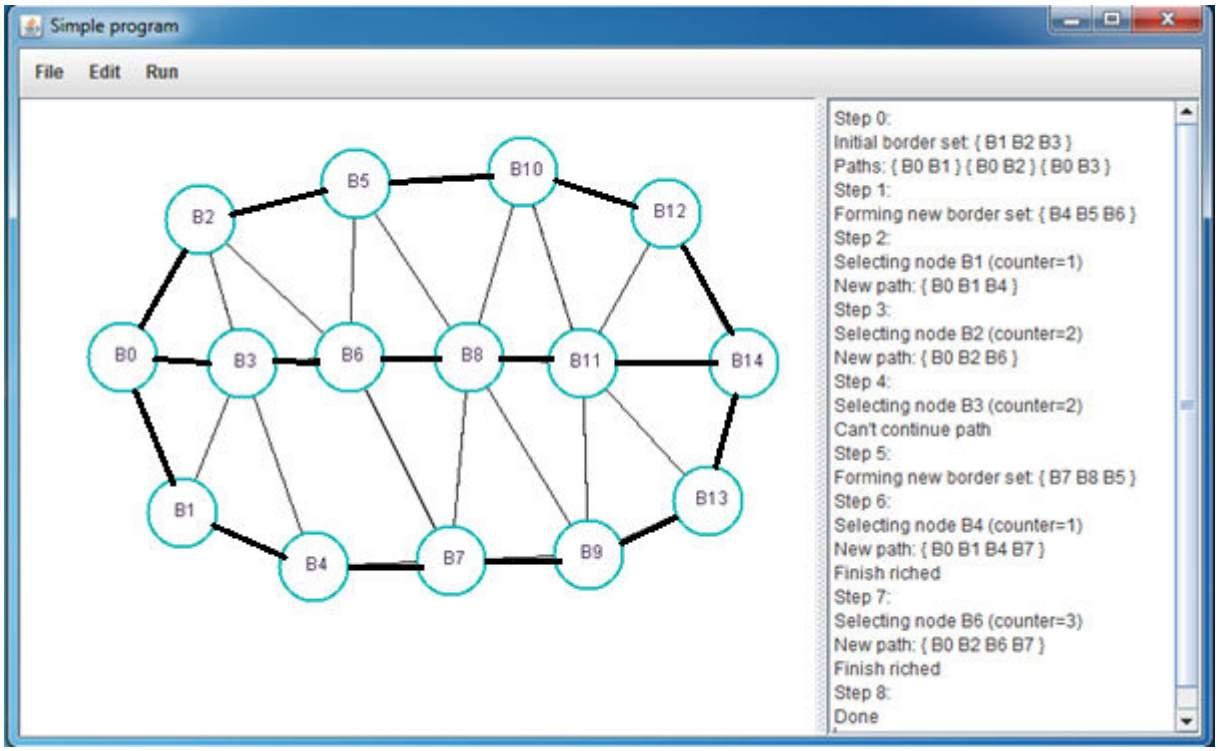


Рис.4.4. Множество путей между вершинами $в_0$ и $в_{14}$

Формирование первого пути { B0 B2 B5 B10 B12 B14 } к вершине $в_{14}$ происходит на 15 шаге из 18 шагов, второго пути { B0 B3 B6 B8 B11 B14 } происходит на 16 шаге, и третьего пути происходит на последнем 18 шаге. Таким образом, все множество путей формируется за один проход алгоритма. На формирование первого пути затрачивается 30 операций, на второго пути еще 2 операции и на третьего пути еще 3 операции, всего 35 операций. В среднем

количество шагов равно числу пройденных вершин, умноженному на их среднюю степень.

Данный алгоритм позволяет сформировать множества путей от одной вершины к нескольким вершинам. Рассмотрим случай формирования путей от вершины $в_0$ множества вершин $\{в_7,в_{10},в_{14}\}$.

На рис. 4.5 представлен пример формирования первого пути от вершины $в_0$ к вершине $в_7$, при этом путь от вершины $в_0$ продлевается до смежных с ней вершин, образуя первый уровень дерева с множеством висячих вершин $\{в_1,в_2,в_3\}$.

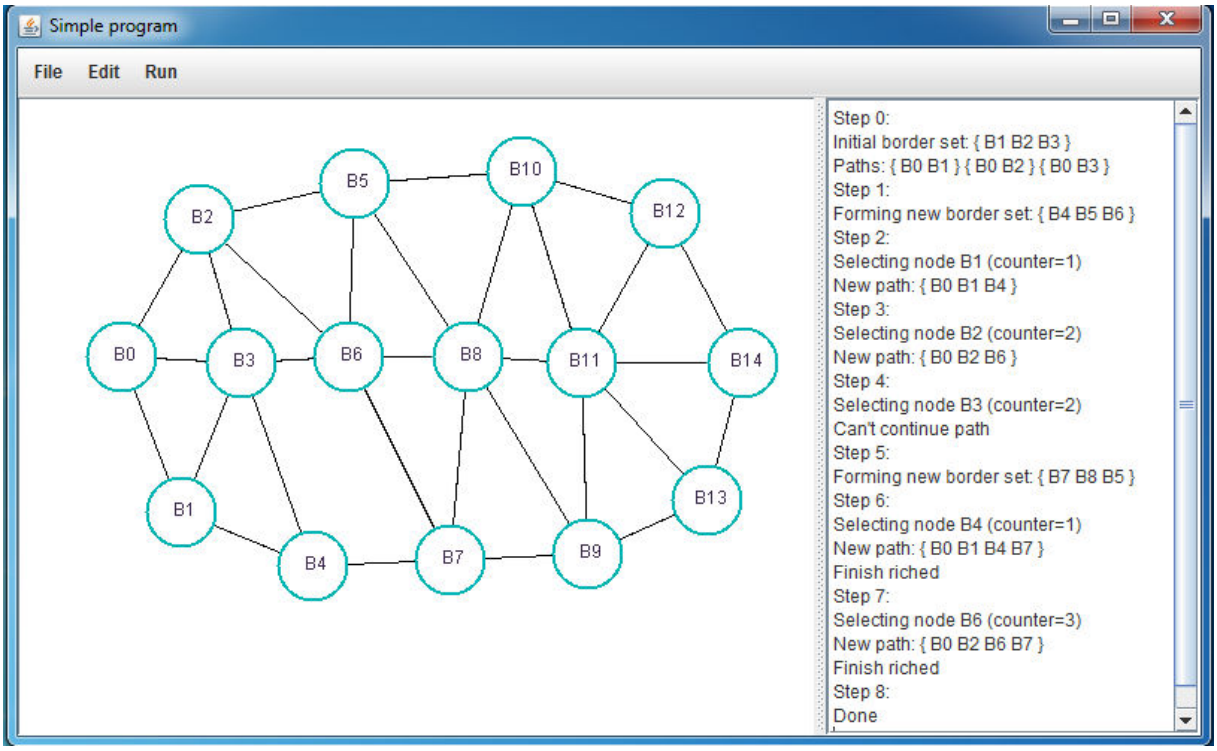


Рис.4.5. Формирование первого пути от вершины $в_0$ к вершине $в_7$

На следующем этапе формируется второй уровень дерева путей с множеством висячих вершин $\{в_4,в_5,в_6\}$. В результате формируется дерево путей, представленное на рис. 4.6. Следует заметить, что на данном этапе пути к вершинам $в_7$, $в_{10}$ и $в_{14}$ совпадают.

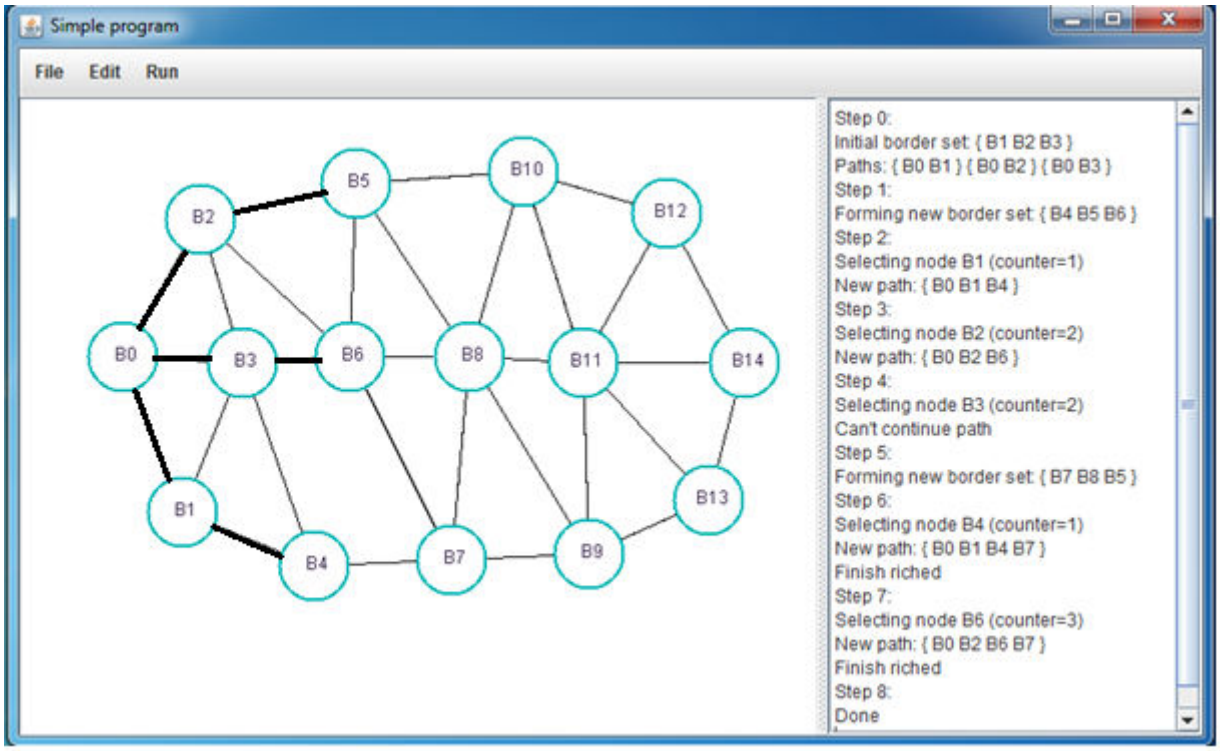


Рис. 4.6. Дерево путей к вершинам $\{в_4,в_5,в_6\}$

Вершины $в_4$ и $в_6$ смежные с вершиной $в_7$, поэтому формируются пути от этих вершин к вершине $в_7$, а путь от вершины $в_5$ продлевается до вершин $\{в_8,в_{10}\}$ (рис. 4.7).

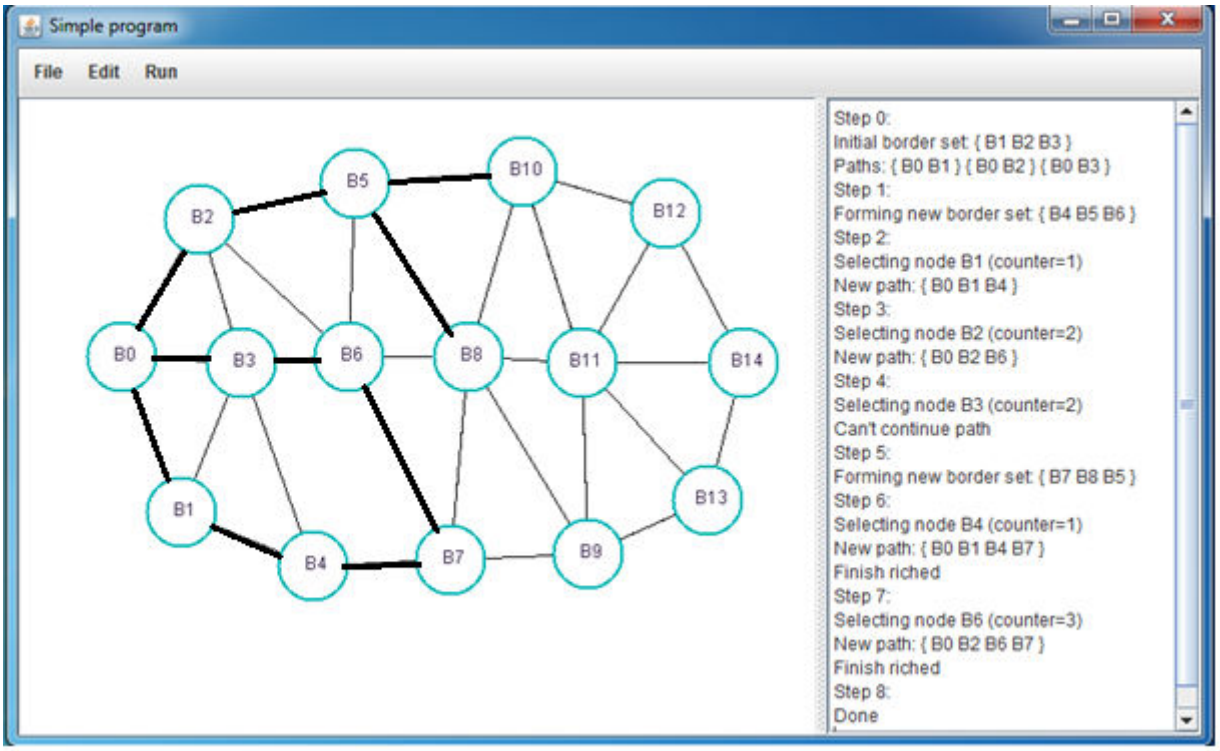


Рис. 4.7. Дерево путей к вершинам $\{в_7,в_8,в_{10}\}$

Формирование всего множества путей между вершинами $в_0$ и $в_7$ заключается в следующем:

Step 0:

Initial border set: { B3 B2 B1 } /*формирование множества граничных узлов для вершины $в_0$ */

Paths: { B0 B3 } { B0 B2 } { B0 B1 } /* формирование путей от вершины $в_0$ к вершинам $в_3$,$в_2$,$в_1$. */

Step 1:

Forming new border set: { B4 B6 B5 } /* новое множество граничных узлов */

Step 2:

Selecting node B1 (counter=1) /* выбор граничной вершины с минимальной значением внешней степени */

New path: { B0 B1 B4 } /* продолжение пути от вершины $в_0$ к вершине $в_4$ */

Step 3:

Selecting node B3 (counter=2) /* выбор граничной вершины с минимальным значением внешней степени */

New path: { B0 B3 B6 } /* продолжение пути от вершины $в_0$ к вершине $в_6$ */

Step 4:

Selecting node B2 (counter=2) /* выбор граничной вершины с минимальным значением внешней степени */

New path: { B0 B2 B5 } /* продолжение пути от вершины $в_0$ к вершине $в_5$ */

Step 5:

Forming new border set: { B10 B7 B8 } /* новое множество граничных узлов */

Step 6:

Selecting node B4 (counter=1) /* выбор граничной вершины с минимальным значением внешней степени */

New path: { B0 B1 B4 B7 } /* сформирован первый путь к вершине $в_7$ */

Step 7:

Selecting node B6 (counter=2) /* выбор граничной вершины с минимальным значением внешней степени */

New path: { B0 B3 B6 B7 } /* сформирован второй путь к вершине $в_7$ */

Step 8:

Selecting node B5 (counter=2) /* выбор граничной вершины с минимальным значением внешней степени */

New path: { B0 B2 B5 B8 } /* продолжение пути от вершины $в_0$ к вершине $в_8$ */

Step 9:

Forming new border set: { B10 B7 B11 B9 } /* новое множество граничных узлов */

Step 10:

Selecting node B8 (counter=4) /* выбор граничной вершины с минимальным значением внешней степени */

New path: { B0 B2 B5 B8 B7 } /* сформирован третий путь к вершине $в_7$ */

Finish riched

Done

В результате работы алгоритма формируются виртуальный канал $с_{0,7}$, состоящий из трех физических каналов (рис. 4.8):

$с_{0,7}=\{L_1=\{в_0,в_1,в_4,в_7\}; L_2=\{в_0,в_3,в_6,в_7\}\ L_3=\{в_0,в_2,в_5,в_8,в_7\}\}$.

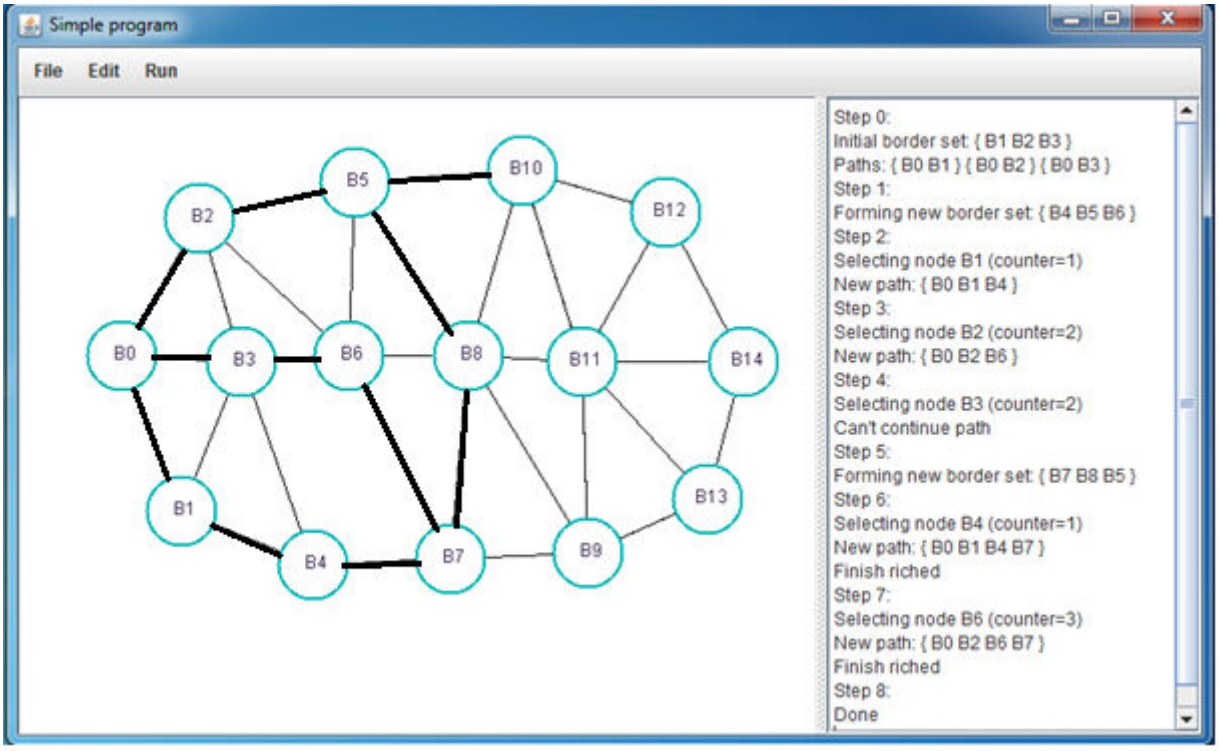


Рис. 4.8. Множество путей от вершины $в_0$ до вершины $в_7$

После формирования пути до вершины $в_7$ (рис.4.8) продолжается формирование множества путей до следующей ближайшей вершины, в данном случае это вершина $в_{10}$.

Step 8:

Selecting node B5 (counter=2) /* граничная вершины с минимальной внешней степенью */

New path: { B0 B2 B5 B10 } /* сформирован первый путь к вершине $в_{10}$ */

Step 9

New path: { B0 B2 B5 B8 } /* продолжение пути от вершины $в_0$ к вершине $в_8$ */

Step 10:

Forming new border set: { B10 B7 B11 B9 } /* новое множество граничных узлов */

Step 11:

Selecting node B8 (counter=4) /* граничная вершины с минимальной внешней степенью */

New path: { B0 B2 B5 B8 B7 } /* сформирован третий путь к вершине $в_7$ */

Step 12:

Forming new border set: { B11 B9 B10 } /* новое множество граничных узлов */

Step 13:

Selecting node B7 (counter=1) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B1 B4 B7 B9 } /* сформирован второй путь к вершине $в_9$ */

Step 14:

Selecting node B8 (counter=3) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B3 B6 B8 B10 } /* сформирован второй путь к вершине $в_{10}$ */

Step 15:

Forming new border set: { B11 B13 } /* новое множество граничных узлов */

Step 16:

Selecting node B9 (counter=2) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B1 B4 B7 B9 B11 } /* новое множество граничных узлов */

Step 17:

Selecting node B11 (counter=2) /* новое множество граничных узлов */

New path: { B0 B1 B4 B7 B9 B11 B10 } /* сформирован третий путь к вершине $в_{10}$ */

Done

В результате работы алгоритма в дополнение к виртуальному каналу $c_{0,7}$ формируются виртуальный канал $c_{0,10}$, состоящий из трех физических каналов (рис. 4.9):

$c_{0,10}=\{L_4=\{в_0,в_1,в_4,в_7,в_9,в_{11},в_{10}\}; L_5=\{в_0,в_3,в_6,в_8,в_{10}\}\ L_6=\{в_0,в_2,в_5,в_{10}\}\}$.

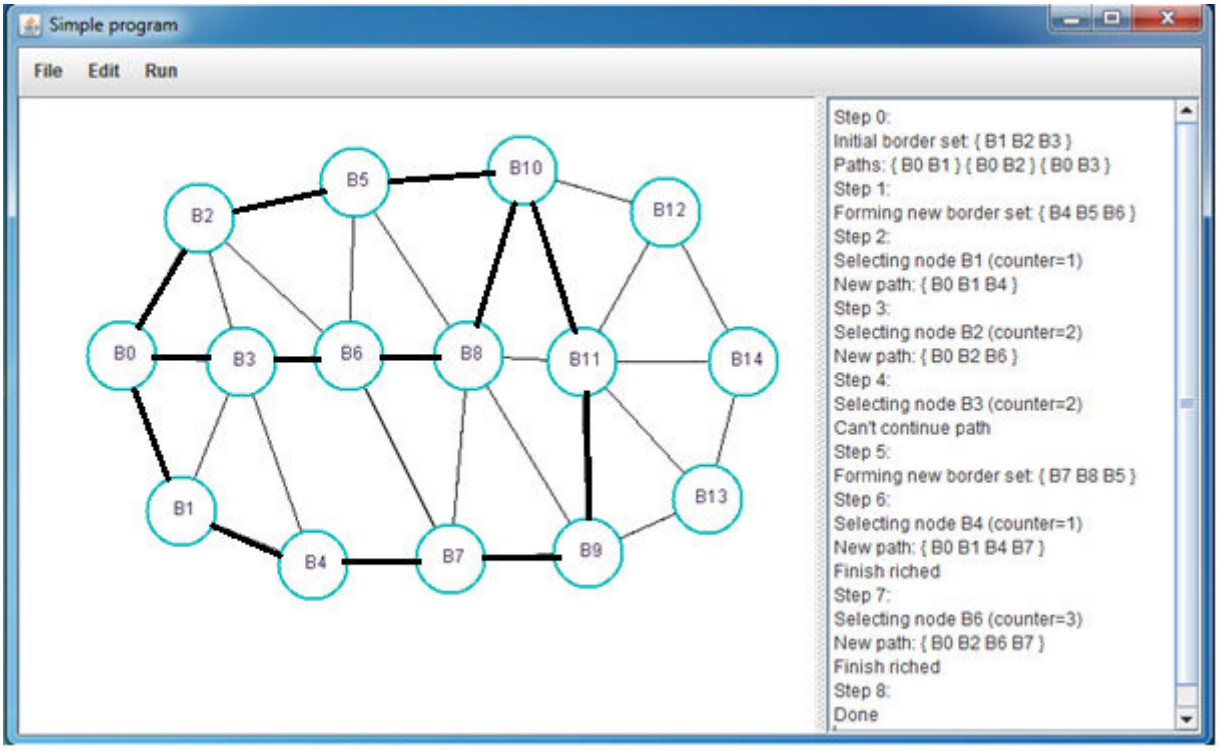


Рис. 4.9. Формирование путей к вершине $в_{10}$

После формирования пути до вершины $в_{10}$ (рис. 4.9) на следующем 18 шаге продолжается формирование множества путей до последней вершины $в_{14}$.

Step 18:

Forming new border set: { B11 B9 B12 } /* новое множество граничных узлов */

Step 19:

Selecting node B8 (counter=2) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B3 B6 B8 B11 } /* сформирован новый путь к вершине $в_{11}$ */

Step 20:

Selecting node B10 (counter=2) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B2 B5 B10 B12 } /* сформирован путь к вершине $в_{12}$ */

Step 21:

Forming new border set: { B13 B14 } /* новое множество граничных узлов */

Step 22:

Selecting node B9 (counter=1) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B1 B4 B7 B9 B13 } /* сформирован путь к вершине $в_{13}$ */

Step 23:

Selecting node B12 (counter=1) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B2 B5 B10 B12 B14 } /* сформирован первый путь к вершине $в_{14}$ */

Step 24:

Selecting node B11 (counter=2) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B3 B6 B8 B11 B14 } /* сформирован второй путь к вершине $в_{14}$ */

Step 25:

Selecting node B13 (counter=1) /* граничная вершина с минимальной внешней степенью */

New path: { B0 B1 B4 B7 B9 B13 B14 } /* сформирован третий путь к вершине $в_{14}$ */

В результате работы алгоритма в дополнение к виртуальным каналам $c_{0,7}$ и $c_{0,10}$ формируются виртуальный канал $c_{0,14}$, состоящий из трех физических каналов (рис. 4.9):

$c_{0,14}=\{L_7=\{в_0,в_1,в_4,в_7,в_9,в_{13},в_{14}\}; L_8=\{в_0,в_3,в_6,в_8,в_{11},в_{14}\}\ L_9=\{в_0,в_2,в_5,в_{10},в_{12},в_{14}\}\}$.

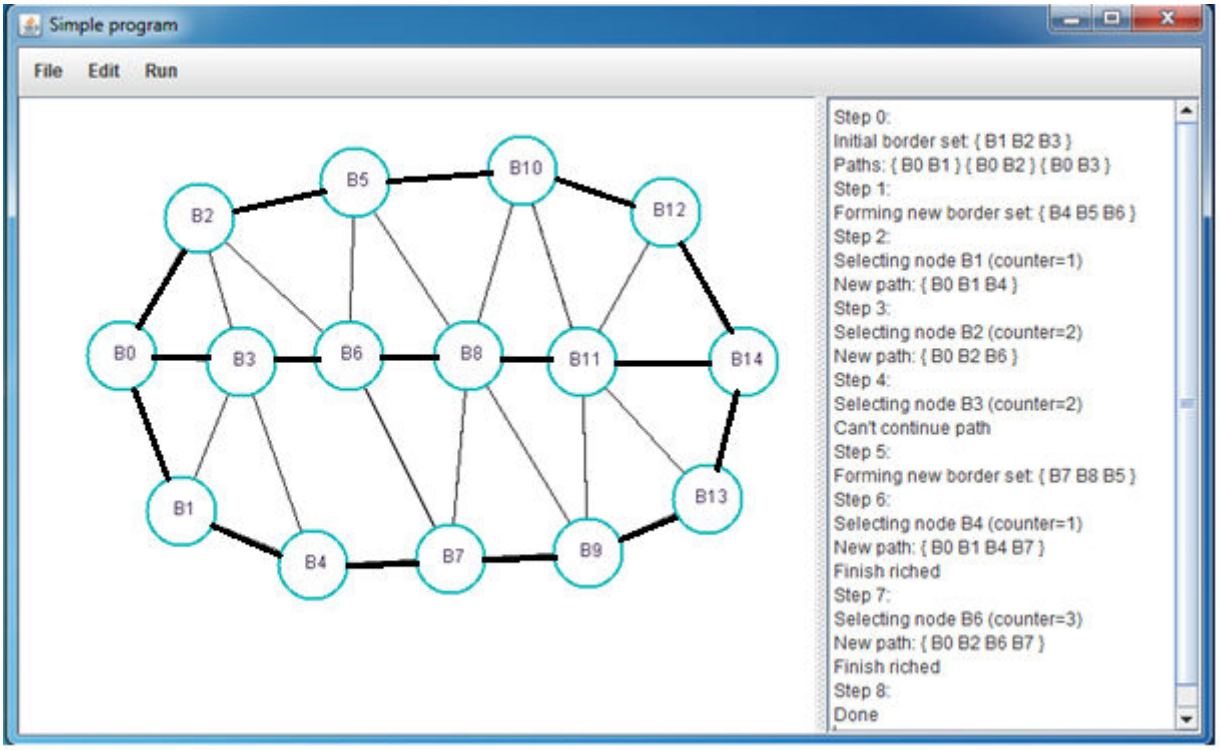


Рис. 4.10. Формирование путей к вершине $в_{14}$

Таким образом, формируется виртуальная звездообразная топологи Grid поверх физической структуры сети (рис. 4.11).

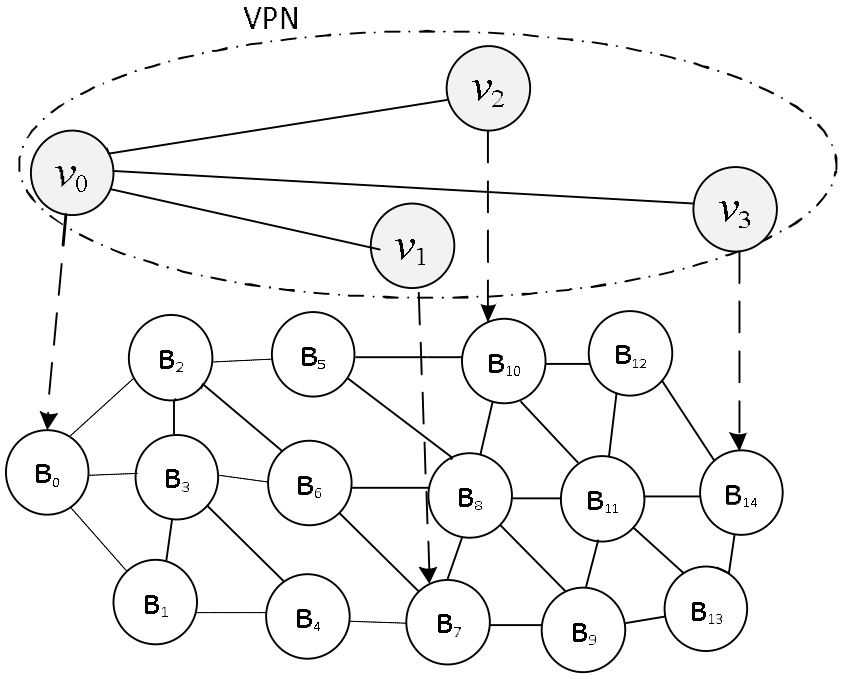


Рис. 4.11. Виртуальная топология Grid системы

4.4. Способ формирования полносвязной структуры виртуальной Grid системы

Для формирования полносвязной топологии необходимо дополнительно сформировать виртуальные пути от вершины $в_7$ к вершинам $в_{10}$ и $в_{14}$, и от вершины $в_{10}$ к вершине $в_{14}$.

Последовательность формирования пути от вершины $в_7$ к вершине $в_{10}$ заключается в следующем:

Step 0:

Initial border set: { B9 B8 B6 B4 } /*формирование множества граничных узлов для вершины $в_7$ */;

Paths: { B7 B9 } { B7 B8 } { B7 B6 } { B7 B4 } /* формирование путей от вершины $в_7$ к вершинам $в_9$, $в_8$, $в_6$, $в_4$. * (рис.4.12) / ;

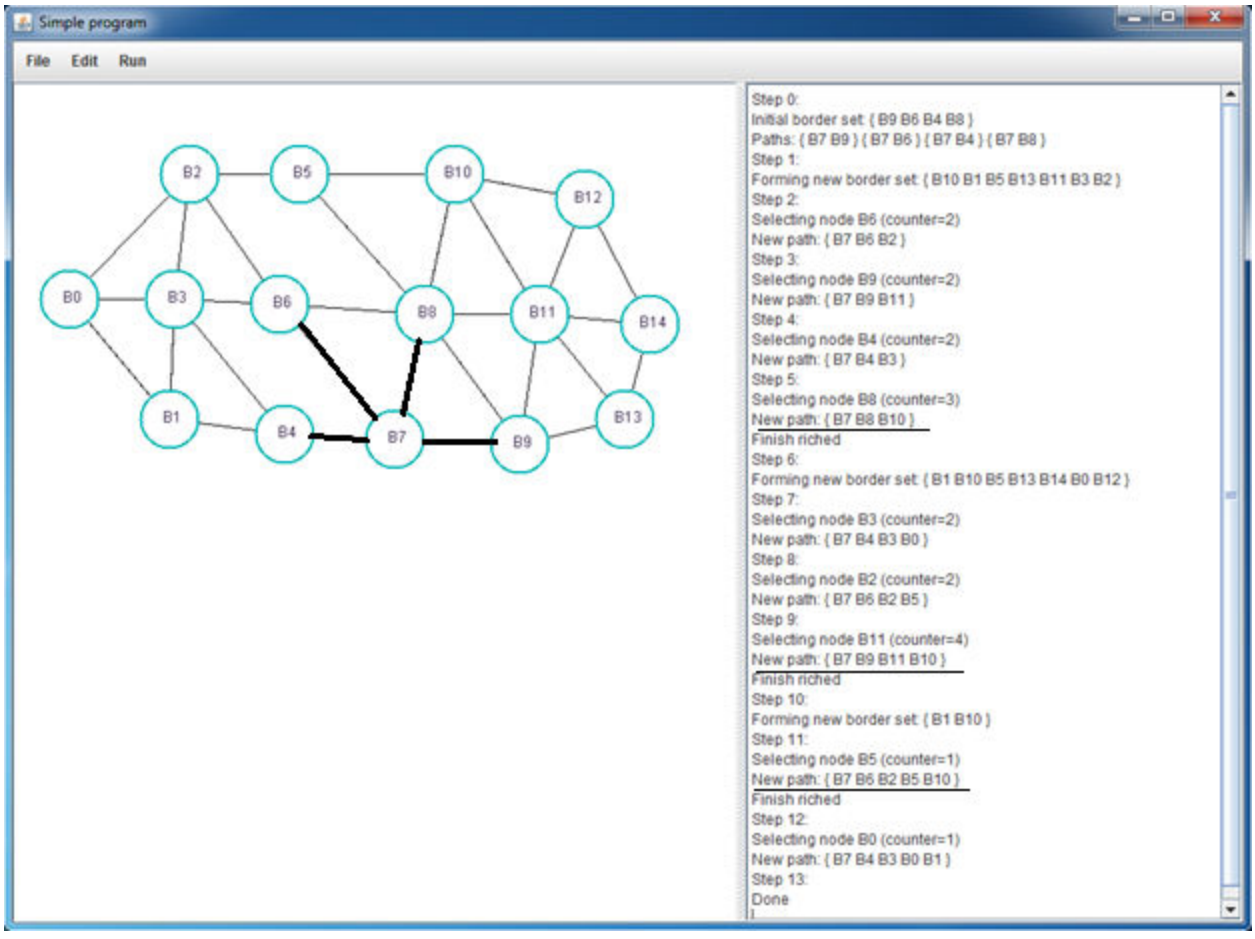


Рис. 4.12. Формирование путей от вершины $в_7$ к вершинам $в_9$,$в_8$,$в_6$,$в_4$

Step 1:

Forming new border set: { B1 B3 B11 B10 B5 B13 B2 } /*формирование множества граничных узлов */;

Step 2:

Selecting node B9 (counter=2) /* выбор граничной вершины с минимальной значением внешней степени */ ;

New path: { B7 B9 B13 } /* продолжение пути { B7 B9 к вершине $в_{13}$ */

Step 3:

Selecting node B6 (counter=2) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B6 B2 } /* продолжение пути { B7 B6} к вершине $в_2$ */

Step 4:

Selecting node B4 (counter=2) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B4 B1 } /* продолжение пути { B7 B4} к вершине $в_1$ */

Step 5:

Selecting node B8 (counter=3)

New path: { B7 B8 B10 }

Finish riched

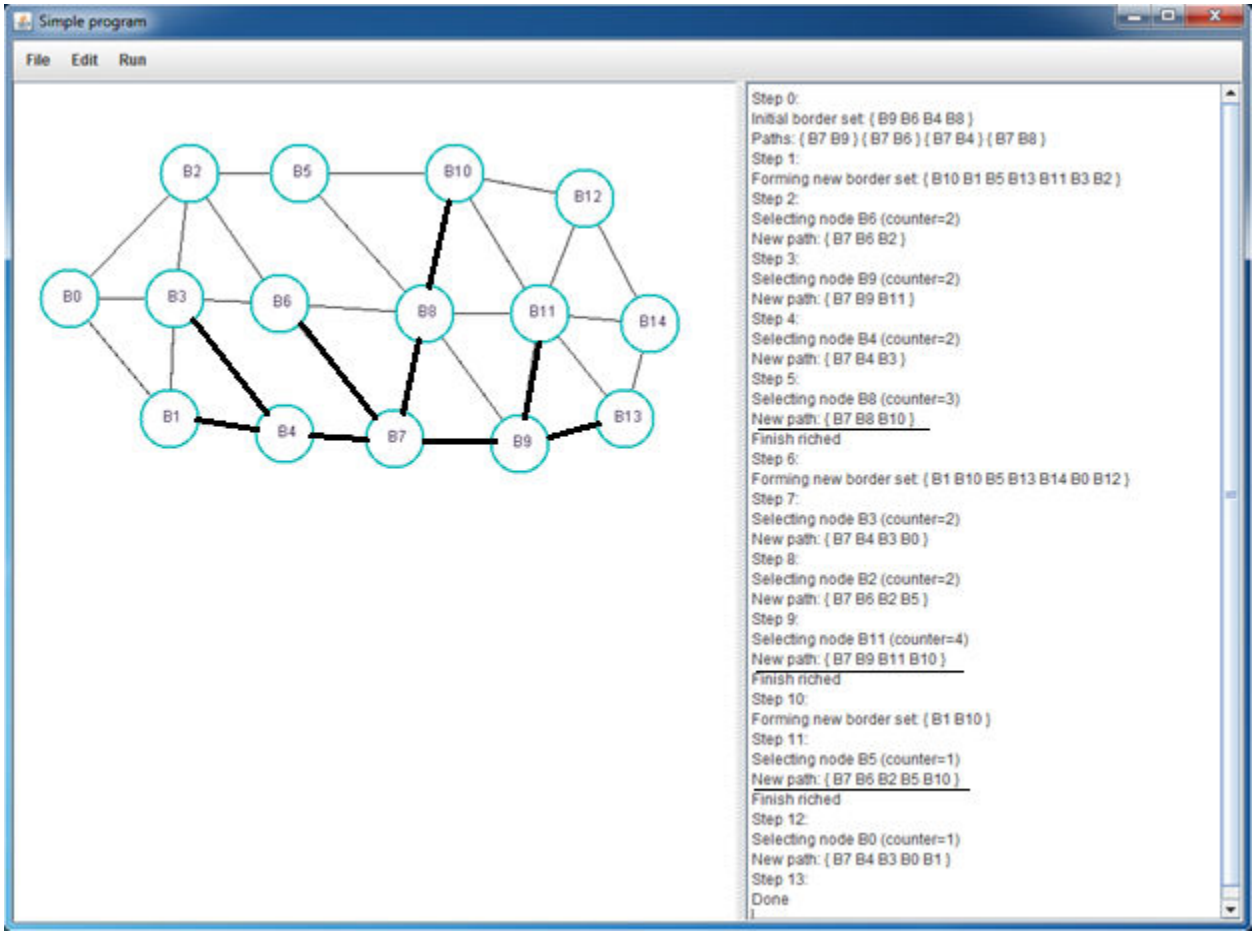


Рис. 4.13. Продолжение путей на втором уровне дерева путей

Затем осуществляется продолжение путей на третьем уровне.

Step 6:

Forming new border set: { B14 B3 B11 B0 B5 } /*формирование множества граничных узлов */;

Step 7:

Selecting node B1 (counter=2) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B4 B1 B0 } /* продолжение пути */

Step 8:

Selecting node B13 (counter=2) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B9 B13 B11 } /* продолжение пути */

Step 9:

Selecting node B2 (counter=3) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B6 B2 B5 } /* продолжение пути */

Step 10:

Forming new border set: { B14 B3 B10 B12 } /*формирование множества граничных узлов */;

Step 11:

Selecting node B0 (counter=1) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B4 B1 B0 B3 } /* продолжение пути */

Step 12:

Selecting node B5 (counter=1) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B6 B2 B5 B10 } /* продолжение пути */

Finish riched

Формируется дерево путей, представленное на рис. 4.14. Step 6:

Forming new border set: { B14 B3 B11 B0 B5 } /*формирование множества граничных узлов */;

Step 7:

Selecting node B1 (counter=2) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B4 B1 B0 } /* продолжение пути */

Step 8:

Selecting node B13 (counter=2) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B9 B13 B11 } /* продолжение пути */

Step 9:

Selecting node B2 (counter=3) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B6 B2 B5 } /* продолжение пути */

Step 10:

Forming new border set: { B14 B3 B10 B12 } /*формирование множества граничных узлов */;

Step 11:

Selecting node B0 (counter=1) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B4 B1 B0 B3 } /* продолжение пути */

Step 12:

Selecting node B5 (counter=1) /* выбор очередной граничной вершины с минимальным значением внешней степени */ ;

New path: { B7 B6 B2 B5 B10 } /* продолжение пути */

Finish riched

Формируется дерево путей, представленное на рис. 4.14.

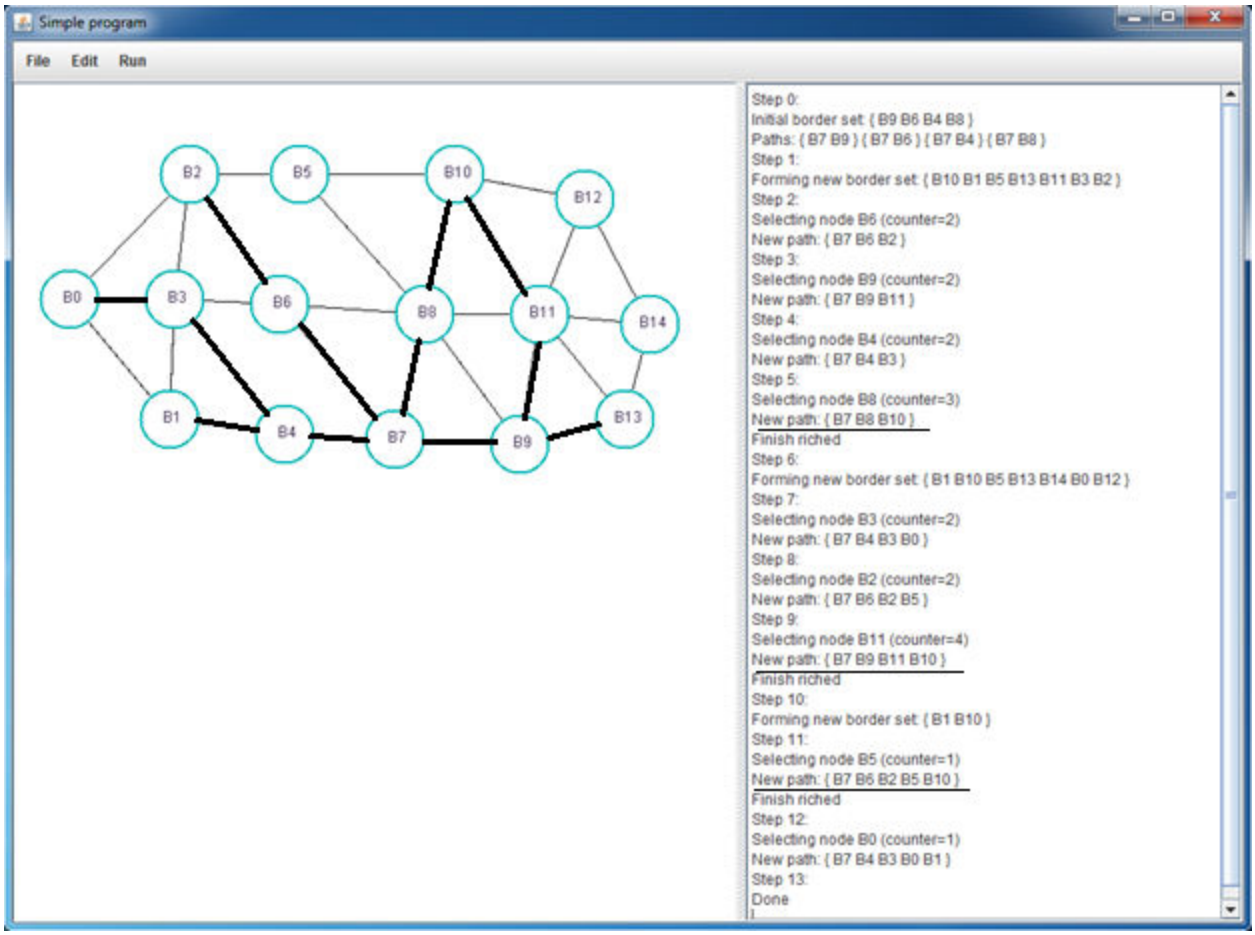


Рис. 4.14. Продолжение путей на третьем уровне дерева путей

Затем осуществляется завершение формирования дерева путей (рис. 4.15).

Step 13:

Selecting node B11 (counter=3)

New path: { B7 B9 B13 B11 B10 }

Finish riched

Step 14:

Done

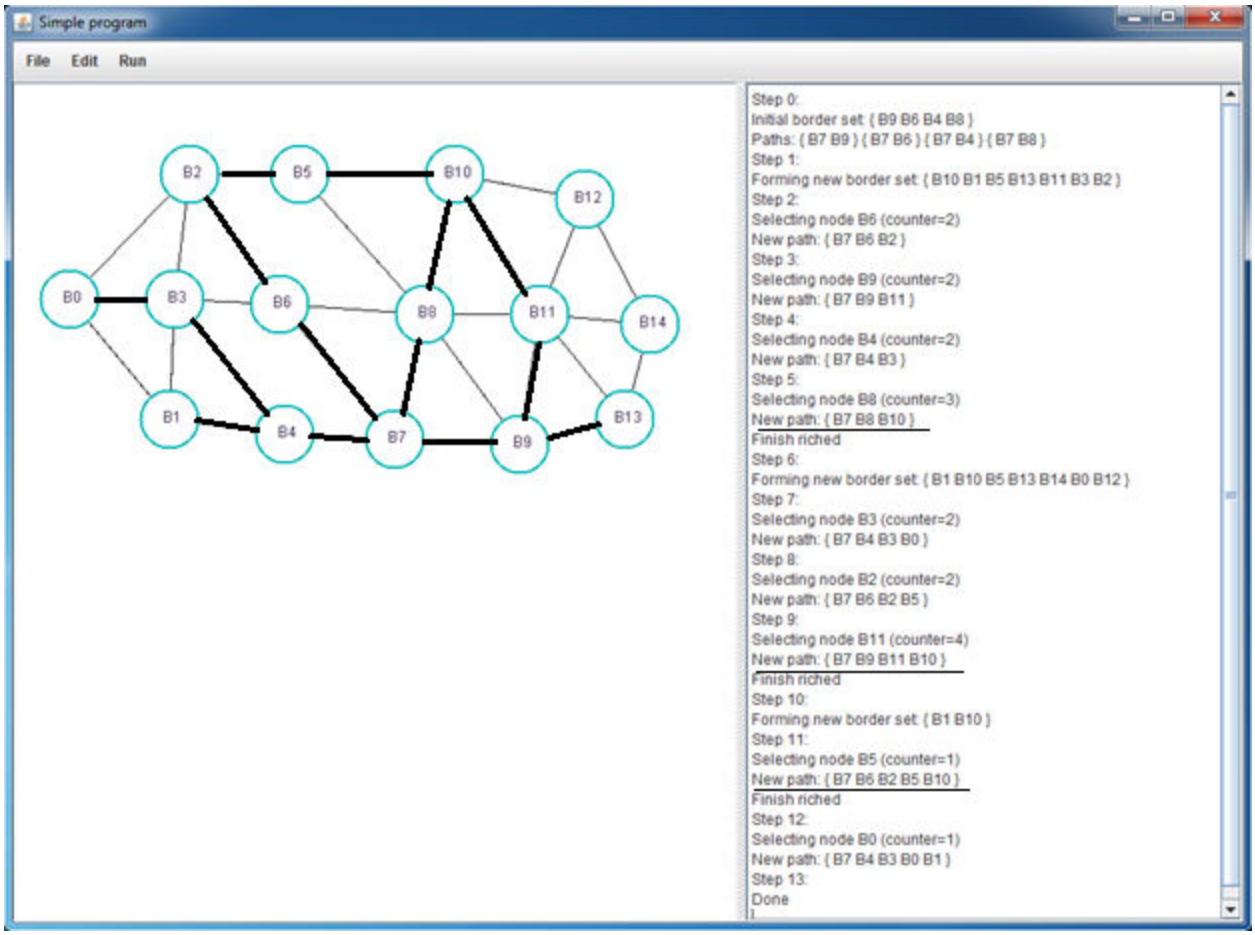


Рис. 4.15. Завершение формирования путей

На рис 4.16 представлен начальный этап формирования множества путей между вершинами $в_{10}$ и $в_{14}$.

Step 0:

Initial border set: { B12 B5 B8 B11 }

Paths: { B10 B12 } { B10 B5 } { B10 B8 } { B10 B11 } .

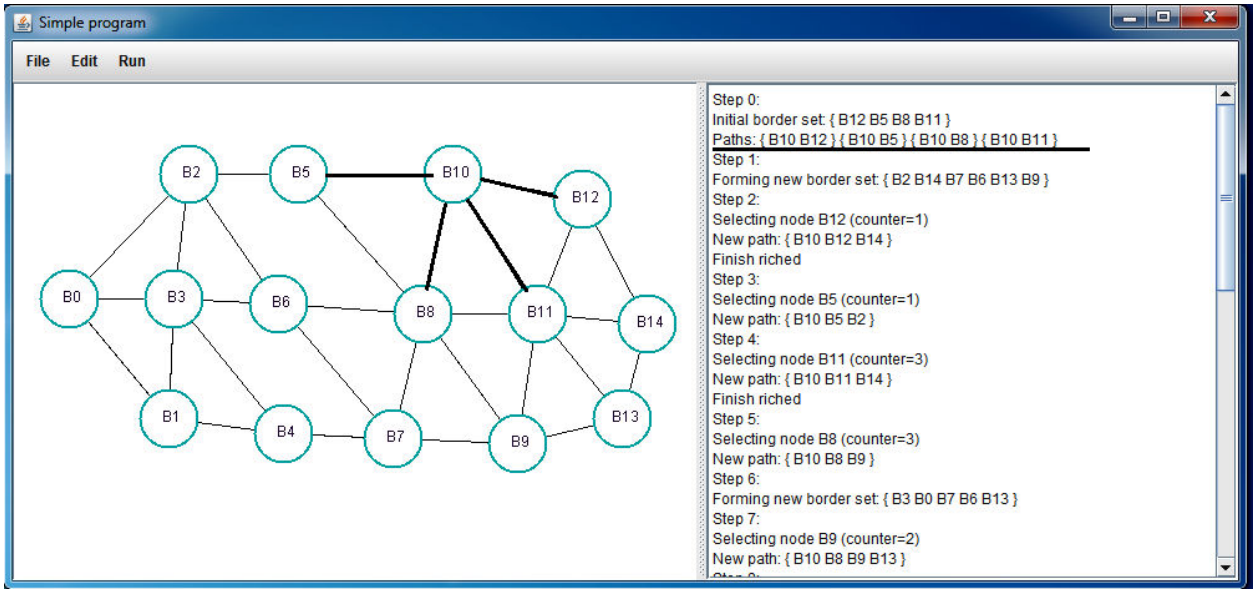


Рис 4.16. Пути к вершинам, смежным с вершиной $в_{10}$

Начальный этап формирования множества путей между вершинами $в_{10}$ и $в_{14}$ (рис. 4.17):

Step 1:

Forming new border set: { B2 B14 B7 B6 B13 B9 }

Step 2:

Selecting node B12 (counter=1)

New path: { B10 B12 B14 }

Finish riched

Step 3:

Selecting node B5 (counter=1)

New path: { B10 B5 B2 }

Step 4:

Selecting node B11 (counter=3)

New path: { B10 B11 B14 }

Finish riched

Step 5:

Selecting node B8 (counter=3)

New path: { B10 B8 B9 }

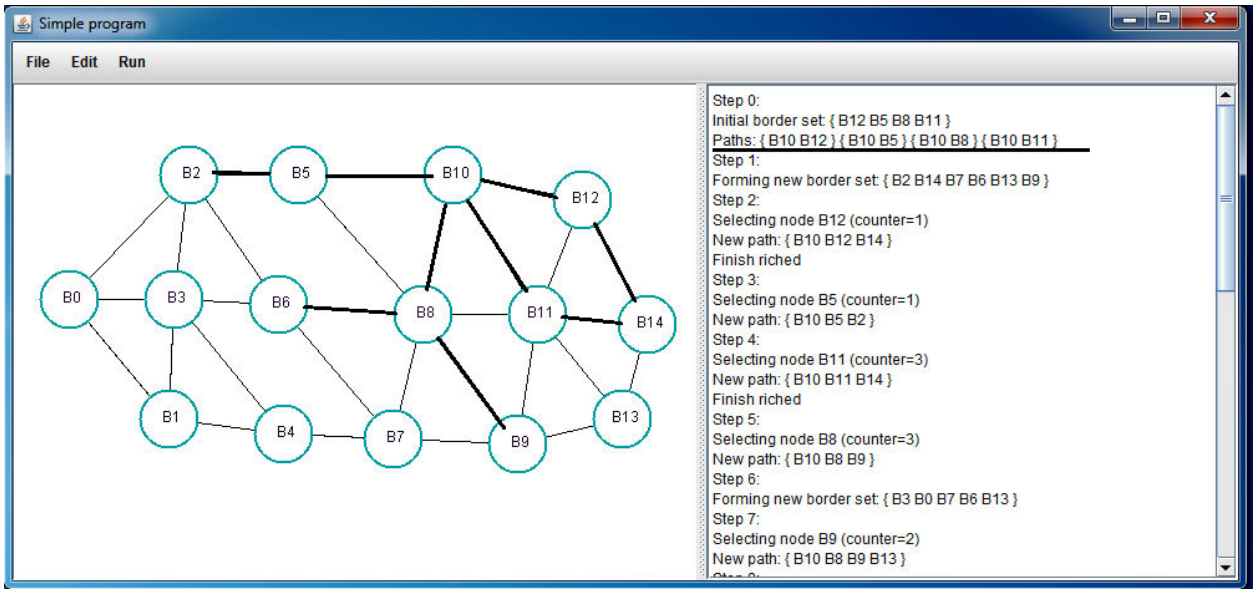


Рис. 4.17. Начальный этап формирования множества путей между вершинами $в_{10}$ и $в_{14}$

На заключительном этапе (рис. 4.18) формируется последний путь между вершинами $в_{10}$ и $в_{14}$:

Step 6:

Initial border set: { B8 B11 B5 B12 }

Paths: { B10 B8 } { B10 B11 } { B10 B5 } { B10 B12 }

Step 7:

Forming new border set: { B9 B13 B2 B14 B7 B6 }

Step 8:

Selecting node B5 (counter=1)

New path: { B10 B5 B2 }

Step 9:

Selecting node B12 (counter=1)

New path: { B10 B12 B14 }

Finish riched

Step 10:

Selecting node B11 (counter=3)

New path: { B10 B11 B14 }

Finish riched

Step 11:

Selecting node B8 (counter=3)

New path: { B10 B8 B6 }

Step 12:

Forming new border set: { B0 B7 B3 }

Step 13:

Selecting node B2 (counter=2)

New path: { B10 B5 B2 B3 }

Step 14:

Selecting node B6 (counter=2)

New path: { B10 B8 B6 B7 }

Step 15:

Forming new border set: { B9 B4 B0 B1 }

Step 16:

Selecting node B7 (counter=2)

New path: { B10 B8 B9 B13 B14 }

Finish riched

Step 17:

Done

В результате между вершинами между вершинами $в_{10}$ и $в_{14}$ формируются следующие пути:

{B10 B12 B14};

{B10 B11 B14};

{B10 B8 B9 B13 B14}.

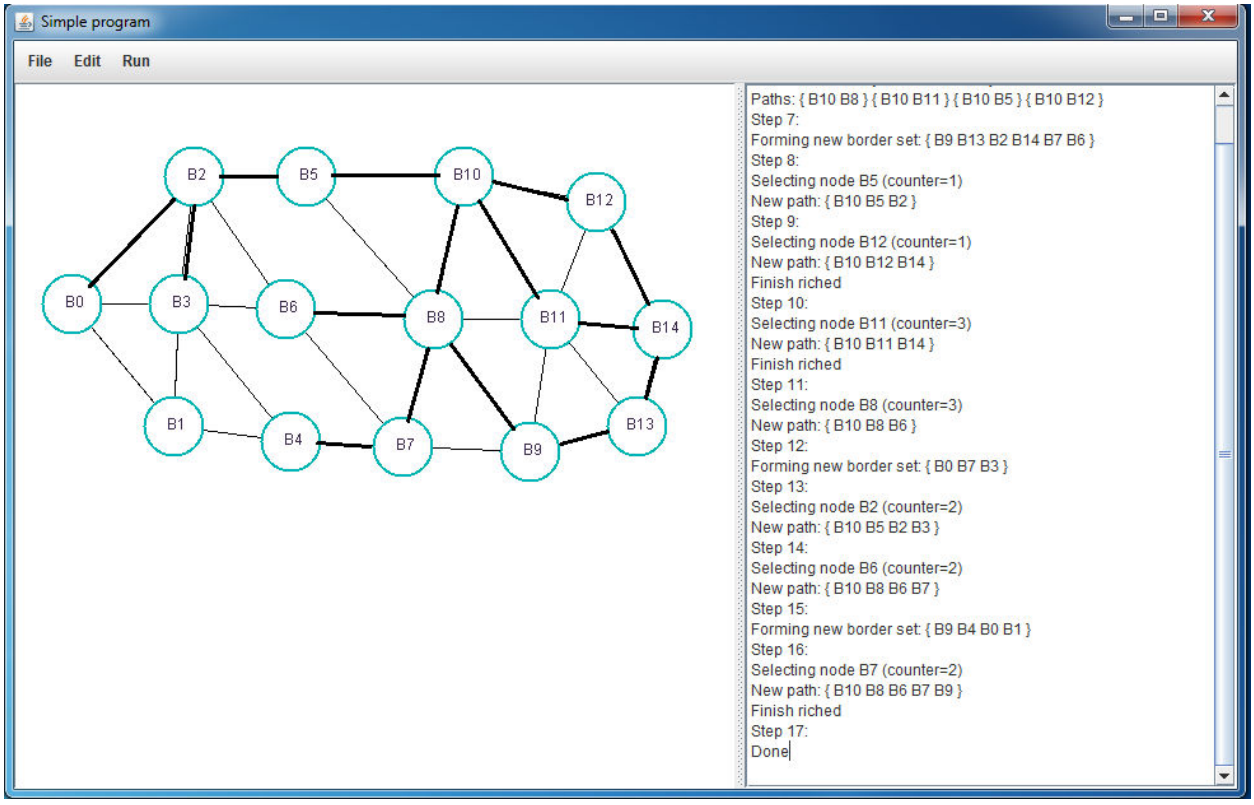


Рис. 4.18. Заключительный этап формирования непересекающихся путей между вершинами $в_{10}$ и $в_{14}$

ВЫВОДЫ ПО ГЛАВЕ 4

В работе предложен и обоснован алгоритм формирования множества путей в DCN. Данный алгоритм является модифицированным алгоритмом поиска в глубину. Предложенный алгоритм максимально учитывает особенности сетевой топологии Fat tree, что способствует более эффективному поиску множества путей передачи данных. В свою очередь применения режима worst-fit позволяет в рамках сетевой топологии Fat tree позволяет обеспечить линейную временную сложность формирования множества путей и обеспечивает лучшую балансировку нагрузки.

ОСНОВНЫЕ ВЫВОДЫ И РЕЗУЛЬТАТЫ РАБОТЫ

В диссертационной работе проведено теоретическое обоснование и получено новое решение задачи формирования максимального множества непересекающихся путей. Решение данной задачи основывается на методах теории графов, теории множеств и комбинаторном анализе. В частности, предложенный в работе способ многопутевой маршрутизации основан на определении множества вершин сочленения деревьев с корневыми вершинами, между которыми формируются непересекающиеся пути. Благодаря этому данный алгоритм характеризуется меньшей временной сложностью по сравнению с известными алгоритмами.

Основные научные и практические результаты заключаются в следующем:

1. На основе теории графов сформулированы критерии наличия максимального количества непересекающихся путей между двумя узлами сети, позволяющие оптимизировать процедуру маршрутизации.

2. Предложены и обоснованы критерии формирования максимально возможного множества непересекающихся путей в компьютерных сетях большой размерности.

3. Разработан способ одновременного формирования множества путей от одного узла к нескольким узлам, позволяющий за счет исключения операции направленного перебора, характерного для комбинаторных алгоритмов существенным образом уменьшить временную сложность формирования виртуальной компьютерной сети.

4. Впервые предложен и обоснован способ формирования непересекающихся путей до вершин минимальной области сочленения подграфов, который позволяет существенным образом уменьшить временную сложность алгоритма формирования непересекающихся и частично пересекающихся путей.

5. На основе метода сочленения деревьев предложен способ формирования множества непересекающихся путей, в качестве корневых вершин которых выступают узлы компьютерной сети, который за счет использования алгоритма «встречной волны» позволяет сократить время и область поиска множества непересекающихся путей.

6. Предложен способ формирования стека меток, обеспечивающих формирования многопутевых туннелей в рамках виртуальных сетей между граничными маршрутизаторами сети MPLS.

7. Предложен алгоритм многопутевой маршрутизации в распределенных центрах данных с сетевой топологии Fat tree, позволяющий за счет свойства самоподобия и учета взаимного расположения вычислительных узлов, между которыми осуществляется обмен информацией, существенным образом сократить время маршрутизации.

8. С практической точки зрения полученные в работе результати позволяют повысить эффективность процедуры управления трафиком в распределенных вычислительных системах большой размерности и оптимизировать процесс доступа к ресурсам в центрах данных.

Список использованных источников литературы

1. Ditixa Vyas. Survey of Distributed Multipath Routing Protocols for Traffic Management / Ditixa Vyas, Ritesh Patel, Amit Ganatra // International Journal of Computer Applications (0975-8887), Volume 63-No.17, February 2013, p. 42-48.

2. Karthiga. S. Traffic Engineering System Based on Adaptive Multipath Routing / Karthiga. S, Balamurugan. M.S // International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 2, February 2013, p. 659-664.

3. Ron Banner. Multipath Routing Algorithms for Congestion Minimization / Ron Banner, Ariel Orda // IEEE/ACM Transactions on Networking (TON), Volume 15, Issue: 2, April 2007, p. 413-424.

4. Jiazi Yi. Multipath optimized link state routing for mobile ad hoc networks Ad Hoc Networks / Jiazi Yi, Asmaa Adnane, Sylvain David Benoît Parrein // Ad hoc Networks, Volume 9, Issue 1, January 2011, p. 28-47.

5. Кулаков Ю. А. Многопутевая маршрутизация в беспроводных сетях / Кулаков Ю.А., Левчук А.В. // Проблеми інформатизації та управління: Зб.наук.пр.– Выпуск 4 (26), К.: Вид-во Нац. авіац. ун-ту «НАУ-друк», 2010, С. 142-147.

6. As'ad Mahmoud. Reliable Multipath Secure Routing In Mobile Computer Networks / As'ad Mahmoud As'ad Alnaser, Kulakov Y.O. // Computer Engineering and Intelligent Systems (IISTE), Volume 4, No. 2, 2013, p. 8-16.

7. Воротников В.В., Кулаков Ю.А. Многопутевая маршрутизация в mesh сетях с использованием резервных слабосвязанных путей / Воротников В.В., Кулаков Ю.А. // Информационные технологии моделирования и управления. Воронеж: Изд-во «Научная книга».– 2014, Том 85, №1, С. 68-77.

8. Кулаков Ю.О. Спосіб організації багатошляхової безпечної маршрутизації в бездротовій мережі *MPLS* / Кулаков Ю.О., Лукашенко В.В., Левчук А.В. // Вісник Національного авіаційного університету, №1 (50), 2012, С. 101-105.

9. Kulakov Y.O. Multipath Routing in Wireless Networks / Kulakov Y.O., As'ad Mahmoud, As'ad Alnaser // Contemporary Engineering Sciences, Volume 5, no. 6, 2012, p. 251-264.

10. Кулаков Ю.А. Разработка и моделирование процесса безопасной многопутевой передачи информации в мобильных сетях / Кулаков Ю.А., Коган А.В., Пирогов А. А. // Вісн. Національного техн. ун-ту України "КПИ": Інформатика, управління та обчислювальна техніка, К.: ТОВ "ВЕК+", Випуск 54, 2011, С. 145-149.

11. Кулаков Ю.А. Организация на основе теории игр многопутевой безопасной передачи информации / Кулаков Ю.А., Лукашенко В.В., Коган А.В. // Реєстрація, зберігання і обробка даних, Том 14, №1, 2012, С. 85-90.

12. Кулаков Ю.А Формирование оптимальных маршрутов в мобильных сетях на основе модифицированного алгоритма Дейкстры / Кулаков Ю.А., Воротников В.В. // Вісн. Національного техн. ун-ту України "КПИ": Інформатика, управління та обчислювальна техніка, К.: ТОВ "ВЕК+", Випуск 56, 2012, С. 13-19.

13. Brijender Kahanwal. The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle / Brijender Kahanwal, Tejinder Pal Singh // International Journal of Latest Research in Science and Technology, Volume 1, Issue 2, July-August 2012, p. 183-187.

14. Sandeep Singh. A survey on Internet Multipath Routing and Provisioning / Sandeep Singh, Tamal Das, Admela Jukan // IEEE Communications Surveys and Tutorials, July 2015, p. 1-20.

15. Jiayue He. From Multiple Decomposition to TRUMP: Traffic Management Using Multipath Protocol / Jiayue He, Martin Suchara, Malayan Bresler, Jennifer Rexford, and Mung Chiang // www.techrepublic.com/whitepapers, April 2008.

16. Saba Ahsan. Multipath RTP: Applying Multipath Communication to Real-time Applications / Saba Ahsan, Jorg Ott, Varun Singh // Faculty of Electronics, Communications and Automation, November 2011, p. 1-57.

17. Hung-Yun Hsieh. pTCP: An End-to-End Transport Layer Protocol for Striped Connections / Hung-Yun Hsieh, Raghupathy Sivakumar // IEEE International Conference on Network Protocols (ICNP), Paris, France, November 2002, p. 1-10.

18. Kultida Rojviboonchai. An Evaluation of Multi-path Transmission Control Protocol (M/TCP) with Robust Acknowledgement Schemes / Kultida Rojviboonchai, Hitoshi Aida // IEICE Transactions on Communications, Volume E87-B, No. 9, September 2004, p. 2699-2707.

19. Keuntae Park. MTCP: A Transmission Control Protocol for Multi-Provider Environment / Keuntae Park, Yongin Choi, Donggook Kim, and Daeyeon Park // IEEE Consumer Communications and Networking Conference (CCNS), February 2006, p. 735-739.

20. Ming Zhang. A transport layer approach for improving end-to-end performance and robustness using redundant paths / Ming Zhang, Junwen Lai, Arvind Krishnamurthy, Larry Peterson, and Randolph Wang // ATEC'04 Proceedings of the annual conference on USENIX Annual Technical Conference, June-July 2004, p. 1-14.

21. Hadassah Pearlyn Nagathota. Design and Implementations of CMT in Real-time / Master Thesis Electrical Engineering, 2015, p. 1-44.

22. Yingyu Wang. Research of NAT traversal based on RTP relay server under mobile internet environment / Yingyu Wang, Shaojun Xu, Jianmin Wang, Ying Xue, Jun Fu, Bo Hu // Computer Science and Network Technology (ICCSNT), 4th International Conference on December 2015, p. 1370-1375.

23. Costin Raiciu. Practical Congestion Control for Multipath Transport Protocols / Costin Raiciu, Damon Wischik, Handley Mark // January 2009, p. 1-13.

24. Juhoon Kim. Multi-source multipath HTTP (mHTTP): a proposal / Juhoon Kim, Yung-Chih Chen, Ramin Khalili, Don Towsley, Anja Feldmann // SIGMETRICS'14, ACM international conference on Measurement and modeling of computer systems 2014, pp. 583-584.

25 Qiuyu Peng. Multipath TCP: Analysis, Design, and Implementation / Qiuyu Peng, Anwar Walid, Jaehyun Hwang, Steven H. Low // ACM Transactions On Networking, Volume 24, Issue 1, February 2016, p.596-609.

26 Jose M. Camacho. BGP-XM: BGP eXtended Multipath for Transit Autonomous Systems / Jose M. Camacho, Alberto Garcıa-Martınez, Marcelo Bagnulo, Francisco Valera // Computer Networks, Volume 57, Issue 13, March 2013, p. 954-975.

27. M. Handley. TCP extensions for multipath operation with multiple addresses / IETF RFC 6824, January 2013.

28. iOS: Multipath TCP Support in iOS 7. [Online]. Available: http://support.apple.com/en-us/HT201373.

29. Xiaomin Chen. Multipath routing in Path Computation Element (PCE): Protocol extensions and implementation / Xiaomin Chen, Yuesheng Zhong, Admela Jukan // Network and Optical Communications (NOC), 18th European Conference on and Optical Cabling and Infrastructure (OC&i), August 2013, p. 75-82.

30. Qioing Zhang. Survivable path computation in pce-based multi-domain networks / Qioing Zhang, Mohammad M. Hasan, Xi Wang, Paparao Palacharla, and Motoyoshi Sekiya // OSA Journal of Optical Communications and Networking, Volume 4, Issue 6, June 2012, p. 457-467.

31. Harvey Newman. OLiMPS: Openflow Link-layer Multipath Switchnig. DOE ASCR NGN PIs Meeting. / Harvey Newman, Artur Barczyk, and Michael Bredel // [Online]. http://www.orau.gov/ngnspi2014.

32. Akira Nagata. Delivering a File by Multipath-Multicast on OpenFlow Networks / Akira Nagata, Yoshiaki Tsukiji, and Masato Tsuru // Intelligent Networking and Collaborative Systems (INCoS) 5th International Conference on IEEE, October 2013, p. 835-840.

33. Scudder J. Advertisement of Multiple Paths in BGP / Scudder J., Retana A., Walton D., and Chen E. // IETF RFC 7911, July 2016.

34. Jose. M. Camacho. BGPXM: BGP eXtended Multipath for Transit Autonomous Systems / Jose. M. Camacho, Alberto Garcıa-Martınez, Marcelo Bag-

nulo, and Francisco Valera // Computer Networks, Volume 57, Issue 4, March 2013, p. 954-975.

35. Stefano Secci. Peering Equilibrium Multipath Routing: a Game Theory Framework for Internet Peering Settlements / Stefano Secci, Jean-Louis Rougier, Achille Pattavina, Fioravante Patrone, Guido Maier // IEEE/ACM Transactions on Networking, Volume 19, Issue 2, April 2011, p. 419-432.

36. Varun Singh. MPRTP: multipath considerations for real-time media / Varun Singh, Saba Ahsan, and Jorg Ott // MMSys'13 in Proceedings of the 4th ACM Multimedia Systems Conference, March 2013, p. 190-201.

37. Mohammad Karim Jafaryan. A New Unicast Routing Algorithm for Load Balancing in Multi-Gateway Wireless Mesh Networks / Mohammad Karim Jafaryan, Mohammad Ali Jabraeil Jamali, Shahin Akbarpour // International Journal of Computer Networks and Communications Security, Volume 1, No. 6, November 2013, p. 251-256.

38 Siguang Chen. Game Theoretic Approach in Multipath Routing for Tradeoff between Routing Security and Performance / Siguang Chen, Meng Wu // Computer Supported Cooperative Work in Design (CSCWD), 14th International Conference, May 2010, p. 717-722.

39 Sebastien Berton. Secure, Disjoint, Multipath Source Routing Protocol (SDMSR) for Mobile Ad-Hoc Networks / Sebastien Berton, Hao Yin, Chuang Lin, Geyong Min // Grid and Cooperative Computing (GCC), Fifth International Conference, December 2006, p. 1-8.

40. Jianxin Liao. Load-Balanced One-hop Overlay Multipath Routing with Path Diversity / Jianxin Liao, Shengwen Tian, Jingyu Wang, Tonghong Li, and Qi Qi // KSII Transactions on Internet and Information Systems, Volume 8, No. 2, February 2014, p. 443-461.

41. Junwei Jin. A Multipath Routing Protocol Based on Bloom Filter for Multihop Wireless Networks / Junwei Jin, Sanghyun Ahn // Mobile Information Systems Volume 2016, p. 1-10.

42. Donghyun Kim. Constructing Minimum Connected Dominating Sets with Bounded Diameters in Wireless Networks / Donghyun Kim, Yiwei Wu, Yingshu Li, Ding-Zhu Du // Transactions on Parallel and Distributed Systems, Volume 20, Issue 2, February 2009, p. 147-157.

43. D. Farinacci, The Locator/ID Seperation Protocol (LISP) / D. Farinacci, V. Fuller, D. Meyer, and D. Lewis // RFC 6830, January 2013.

44. Mohamed Amine. A Multipath Lifetime-Prolonging Routing Algorithm for Wireless Ad Hoc Networks / Mohamed Amine, Karim Tamine // International Journal of Advanced Computer Science and Applications, Volume 7, Issue 1, 2016, p. 501-511.

45. Шувалов В.П. Классификация методов многопутевой маршрутизации / Шувалов В.П., Вараксина И.Ю. // T-Comm – Телекоммуникации и транспорт, Том №8, №1, 2014, С. 29-32.

46. Sheng Huang. Multipath Provisioning in WDM Networks. / Sheng Huang, Biswanath Murkherejee // International Conference on Communications (ICC’08), May 2008, p. 5300-5304.

47. Smita Rai. Reliable Multi Path Provisioning for High Capacity Backbone Mesh Network / Smita Rai, Omkar Deshpande, Canhui Ou, Charles U. Martel, and Biswanath Mukherjee // ACM Transactions on Networking, Volume 15, Issue 4, p. 803-812.

48. Ananya Das. New Approach to Reliable Multipath Provisioning / Ananya Das, Charles Martel, Biswanath Mukherjee, and Smita Rai // OSA Journal of Optical Communication Networks, Volume 3, Issue 1, January 2011, p. 95-103.

49. Ananya Das. Partial Protection Approach Using Multipath Provisioning / Ananya Das, Charles Martel, Biswanath Mukherjee // International Conference on Communications (ICC'09), August 2009, p. 1256-1260.

50. Lei Song. On the Study of Multiple Backups and Primary Backup Link Sharing for Dynamic Service Provisioning in Survivable WDM Mesh Networks / Lei Song, Biswanath Mukherjee // Journal on Selected Areas in Communications, Volume 26, Issue 6, August 2008, p. 84-91.

51. Гуровиц В.М. Алгоритм поиска в глубину. [Електронний ресурс] – Информатика (Теория графов), 2014. – Режим доступу: http://foxford.ru/wiki/informatika/algoritm-poiska-v-glubinu.

52. Canfeng Chen. Multipath Routing Modeling in Ad Hoc Networks / Canfeng Chen, Weiling Wu Zheng Li // International Conference on Communications (ICC'09), August 2005, p. 2594-2598.

53. Cai Ling. Load Balancing Algorithm Based on Time Series Prediction of Packet Loss / Cai Ling, Jinkuan Wang, Cuirong Wang, Xu Peng // 4th Conference on Industrial Electronics and Applications (ICIEA), June 2009, p. 145-149.

54. Shaohua Liu. A Dynamic Traffic Distribution Strategy for Multipath Routing / Shaohua Liu, Xu Zhang, Junsheng Yu, Yinghui Liu, Xiaodong Chen // 7th International Conference on Information, Communications and Signal Processing (ICICS'09), January 2010, p. 1256-1260.

55. Medhi Deepankar, Ramasamy Karthikeyan. Network Routing: Algorithms, Protocols, and Architectures. Morgan Kaufmann Publishers, 2007. 788 p.

56. Lemeshko A.V. Research on Tensor Model of Multipath Routing in Telecommunication Network with Support of Service Quality by Greate Number of Indices / Lemeshko A.V., Evseeva O.Yu., Garkusha S.V. // Telecommunications and Radio Engineering, Volume 73, No. 15, January 2014, p. 1339-1360.

57. Mahalakshmi C., Ramaswamy M. Multipath Data Transfer Scheme for Virtual Private Networks // International Journal of Computer Applications (IJCA), Volume 44, No.8, April 2012, p. 27-31.

58. Jingyu Wang. Estimating the Effects of Multipath Selection on Concurrent Multipath Transfer / Jingyu Wang, Jianxin Liao, Jing Wang, Tonghong Li, and Qi Qi // KSII Transactions on Internet and Information Systems, Volume. 8, No. 4, April 2014, p. 1406-1423.

59. T. Benson. Network Traffic Characteristics of Data Centers in the Wild / T. Benson, A. Akella, and D. A. Maltz // 10th ACM SIGCOMM conference on Internet Measurement, November 2010, p. 267-280.

60. Eun-Sung Jung. Distributed Multipath Routing Algorithm for Data Center Networks / Eun-Sung Jung, Venkatram Vishwanath, Rajkumar Kettimuthu // International Workshop on Data Intensive Scalable Computing Systems (DISCS), November 2014, p. 49-56.

61. S. Nash, A. Sofer. Linear and Nonlinear Programming, ser. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 1996. [Online]. Available: http://books.google.com/ books?id=MQAoAQAAMAAJ

62. Omair Fatmi. Distributed Multipath Routing for Data Center Networks Based on Stochastic Traffic / Omair Fatmi, Deng Pan // 11th International Conference on Networking, Sensing and Control (ICNSC), April 2014, p. 536-541.

63 Chuanxiong Guo. Bcube: a high performance, server-centric network architecture for modular data centers / Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu // Proceedings of the ACM SIGCOMM conference on Data Communication, Volume 39, Issue 4, October 2009, p. 63-74.

64. Y. Li. OpenFlow based load balancing for Fat-Tree networks with multipath support / Y. Li, and Deng Pan // Proceedings 12th International Conference on Communications (ICC'13), June 2013, p. 1-5.

65. Costin Raiciu. Improving Datacenter Performance and Robustness with Multipath TCP / Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, Mark Handley // Proceedings of the ACM SIGCOMM Computer Communication Review, Volume 41, Issue 4, August 2011, p. 266-277.

66. Luo Shouxi. Traffic-aware WDC Embedding in Data Center: A Case Study of Fattree / Luo Shouxi, Yu Hongfang, Li Lemin, Liao Dao, Sun Gang // China Communications, Volume 11, Issue 7, July 2014, p. 142-152.

67. Wenzhi Cui. DiFS: Distributed Flow Scheduling for Adaptive Routing in Hierarchical Data Center Networks / Wenzhi Cui, Chen Qian // ACM Symposium on Architectures for Networking and Communications Systems (ANCS), October 2014, p. 53-64.

68. Fung Po Tso. Improving data center network utilization using near-optimal traffic engineering / Fung Po Tso, and Dimitros P. Pezaros // Parallel and Distributed Systems, Volume 24, Issue 6, June 2013, p. 1139-1148.

69. Eric Jo. A Simulation and Emulation Study of SDN-based Multipath Routing for Fat-Tree Data Center Networks / Eric Jo, Deng Pan, Jason Liu, Linda Butler // Proceedings of the Winter Simulation Conference, July 2014, p. 3072-3083.

70. Shafi'i Muhammad Abdulhamid. On-Demand Grid Provisioning Using Cloud Infrastructures and Related Virtualization Tools: A Survey and Taxonomy / Shafi'i Muhammad Abdulhamid, Muhammad Shafie Abd Latiff, Mohammed Bakri Bashir // International Journal of Advanced Studies in Computer Science and Engineering (IJASCSE), Volume 3, Issue 1, February 2014, p. 49-59.

71. N. Mohan Krishna Varma. Extending Grid Infrastructure Using Cloud Computing / N. Mohan Krishna Varma, and Eunmi Choi // Ubiquitous Information Technologies and Applications, Volume 2014, November 2012, p. 507-516.

72. Кулаков Ю.А. Способ формирования структуры виртуальной Grid-системы / Кулаков Ю.А., Олещенко А.Ю.// Вісник університету «Україна» №2, 2011. С.71-75.

73. Hamada Alshaer. An Overview of Network Virtualization and Cloud Network as a Service / International Journal of Network Management 1 December 2014, in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nem.1882.

74. Faizul Bari. Data center network virtualization: a survey / Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Golam Rabbani, Qi Zhang, Mohamed Faten Zhani // Communications Surveys and Tutorials, Volume 15, Issue 2, September 2012, p. 909-928.

75. Albert Greenberg. VL2: a scalable and flexible data center network. / Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, Sudipta Sengupta //Proceedings of the ACM SIGCOMM conference on Data communication, August 2009, p. 51-62.

76. Chuanxiong Guo. SecondNet: a data center network virtualization architecture with bandwidth guarantees / Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, Yongguang Zhang //, 2010, p. 1-15.

77. Radhika Niranjan Mysore. A.Portland: a scalable fault-tolerant layer 2 data center network fabric / Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, Amin Vahdat. // ACM SIGCOMM Computer communication, Review, Volume 39, Issue 4, October 39-50.

78. Rodrigues H, Turner Y, Santos JR, Victor P, Guedes D. Gatekeeper: supporting bandwidth guarantees for multi-tenantdatacenter networks. In *Proceedings of the 3rd Conference on I/O Virtualization (WIOV)*, Berkeley, CA, USA, 2011; 6–6.

79. Benson T, Akella A, Shaikh A, Sahu S. CloudNaaS: a cloud networking platform for enterprise applications,2011, pp. 1–8.

80. Ballani H, Costa P, Karagiannis T, Rowstron A. Towards predictable datacenter networks, 2011

81. Mudigonda J, Yalagandula P, Mogul J, Stiekes B, Pouffary Y. Netlord: a scalable multi-tenant network architecture forvirtualized datacenters. SIGCOMM Computer Communication Review 2011; **41**(4): 62–73

82. Shieh A, Kandula S, Greenberg A, Kim C, Saha B. Sharing the data center network. In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, Berkeley, CA, USA, 2011.

83. Stephen Kent. Secure Border Gateway Protocol (S-BGP) – Real World Performance and Deployment Issues/ Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo// BBN Technologies: Appeared in Proceedings of the Network and Distributed System Security Symposium (NDSS 2000), February 2000. – San Diego, California, 2000. – p.1-14.

84 Диброва М.А. Способы организации многопутевой маршрутизации / М.А. Диброва, А.В. Коган, Р.М. Капорин // Вісник НТУУ «КПІ». Інформати-

ка, управління та обчислювальна техніка: збірник наукових праць. – Київ: Век+, – 2014. №64С. 21-26.

85 Диброва М.А. Способ конструирования трафика в grid системах / Диброва М.А. Коган А.В. Куценко В. А. // Вісник НТУУ "КПІ". Інформатика, управління та обчислювальна техніка: збірник наукових праць. – Київ: Век+, – 2014. – Реферується наукометричними базами даних: Російський індекс наукового цитування (РІНЦ), *Directory of Open Access Journals (DOAJ), Google Scholar*

86. Кулаков Ю.А. Способ формирования множества непересекающихся путей от одной к нескольким вершинам в компьютерной сети большой размерности / Кулаков Ю.А., Коган А.В., Диброва М.А// Сборник трудов XVI международной научной конференции "Интеллектуальный анализ информации ИАИ-2016" им. Таран Т.А. -Київ: Просвіта – 2016.- С.104-107.

87. Кулаков Ю. А. Формирование множества непересекающихся путей в компьютерныых сетях с применением алгоритма «обратной волны»/ Кулаков Ю. А., Коган А. В., Диброва М. А, Чхаидзе Д. М. // Вісник национального авіаційного університету. – 2015. – №2(51). – С.120-126.

88 E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture, Available: http://www.ietf.org/rfc/rfc3031.

89 R. Braden, et al. "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification," Available: http://www.ietf.org/rfc/rfc2205.

90. RFC 3036 LDP Specification. L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas. January 2001 *http://www.ietf.org/rfc/rfc3036.txt*

91. Диброва М.А. Формирование множества непересекающихся путей между граничными маршрутизаторами сети MPLS / Диброва М.А., Коган А.В., Кулаков Ю. А. // Electronics and Communications. Електроніка та зв'язок. Електроніка та зв'язок. – 2016. – Том 21. – №1(90), – С.50-55.

92. M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in Proceedings of the ACM SIGCOMM 2008 Con-

ference on Data Communication. New York, NY, USA: ACM, 2008, p. 6374. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967

93. Mysore R. N. Portland: a scalable fault-tolerant layer2 data center network fabric. [Text] / Mysore, R. N., A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat // ACM SIGCOMM Computer Communication Review – SIGCOMM '09 – ACM New York, NY, USA, 2009. – URL: http://dl.acm.org/citation.cfm?id=1592575.

94. J. Kim, W. J. Dally, S. Scott, and D. Abts, “Technology-driven, highly-scalable dragonfly topology,” in Proceedings of the 35th Annual International Symposium on Computer Architecture, ser. ISCA ’08. Washington, DC, USA: IEEE Computer Society, 2008, p. 7788. [Online]. Available: http://dx.doi.org/10.1109/ISCA.2008.19

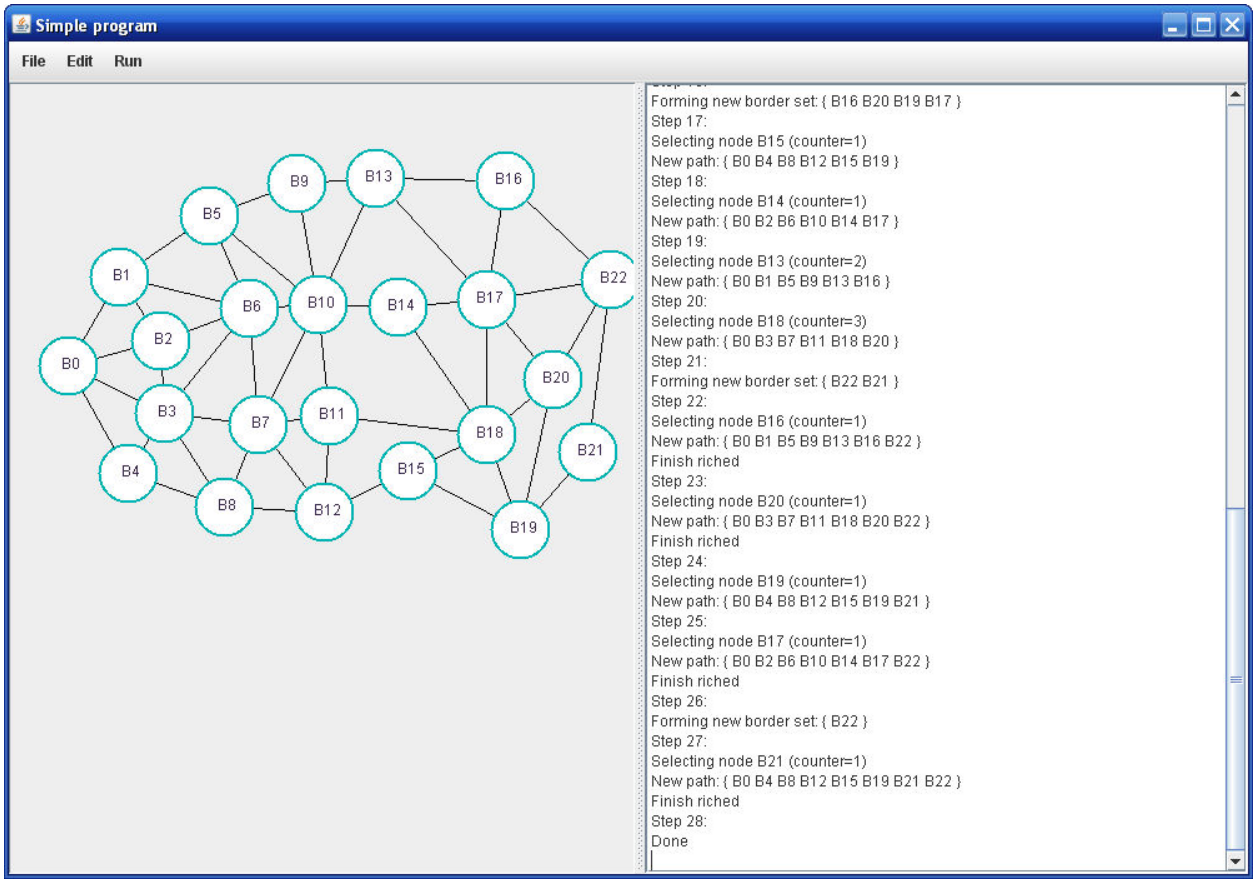
95. Кулаков Ю.А. Разработка и моделирование процесса безопасной многопутевой передачи информации в мобильных сетях / Кулаков Ю.А., Коган А.В., Пирогов А.А. // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка: збірник наукових праць. – К.: Век+, 2011. – № 54. – С. 145-149.

96. Диброва, М.А. Способ формирования множества путей в сетевых центрах данных // М.А. Диброва, А.В. Коган, А.Л. Воробьева // // Вісник НТУУ “КПІ”. Інформатика, управління та обчислювальна техніка: збірник наукових праць. – Київ: Век+, – 2014.

ПРИЛОЖЕНИЯ

Приложение 1. Интерфейс программы и пример формирования множества путей

Интерфейс программы



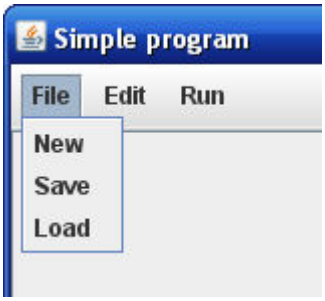
В верхней части есть главное меню. Основное поле слева занимает редактор графов. Справа расположен отчет.

Главное меню

Оно содержит следующие меню:

1. **File** – сохранение/загрузка графа
2. **Edit** – переключение режимов изменения графа
3. **Run** – выполнение моделирования алгоритма

Меню File в свою очередь содержит следующие пункты:



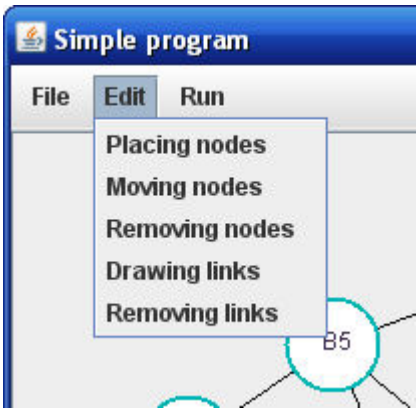
New – очищает рабочую область редактора, все несохраненные данные теряются.

Save – сохранение графа в файл, вызывает вспомогательное окно, в котором можно выбрать каталог и имя сохраняемого файла.

Load – загрузка графа из файла, вспомогательное окно просит указать требуемый файл.

Если при загрузке возникает ошибка (например, файл не существует), то ее краткое описание отобразится в отчете справа.

Меню Edit содержит:



Placing nodes – режим добавление новых вершин графа. Для создания новой вершины нужно кликнуть левой кнопкой мыши в области редактора. Вершина автоматически получит порядковый номер.

Moving nodes – режим перемещение вершин графа. Эта функция имеет значение только для визуального расположения вершин. Вершины можно просто перетягивать на нужное место.

Перемещение может быть немножко дерганным, потому что редактор пытается их выравнивать по сетке, что может быть удобно при необходимости расположить несколько вершин вдоль вертикальной или горизонтальной линии.

Этот режим используется по умолчанию и в него можно перейти не только по соответствующему пункту меню, но и двумя последовательными кликами правой кнопки мыши на пустом месте в области редактора.

Removing nodes – режим удаления вершин. Достаточно просто кликнуть по нужной вершине левой кнопкой мыши. При удалении вершины порядковые номера остальных вершин могут сдвинуться.

Drawing links – режим создания связей между вершинами. Для создания связи нужно последовательно выбрать две вершины левой кнопкой мыши. Первая выбранная вершина запоминается, так что при создании нескольких связей повторно указывать на исходную вершину не нужно. Для сброса выбранной вершины достаточно кликнуть правой кнопкой мыши на пустом месте.

Также в этом режиме рисуется линия, подсказывающая какую вершину мы пытаемся соединить.

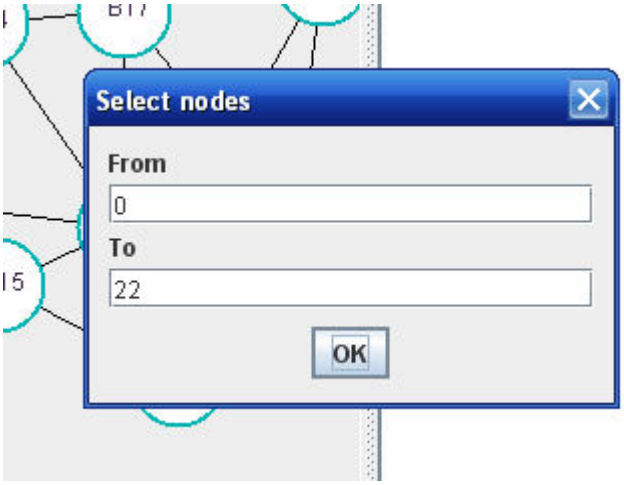
Removing links – режим удаления связей. Работает аналогично предыдущему, только вместо создания связи между выбранными вершинами выполняется удаление (если связь существует).

Таким образом для создания графа необходимо:

1. Выбрать режим Placing nodes и добавить необходимое количество вершин.
2. Выбрать режим Drawing links и для каждой вершины создать ее связи с другими вершинами.
3. Выбрать режим Moving nodes (или кликнуть два раза правой кнопкой мыши на пустом месте в области редактора) и расположить вершины удобным образом.
4. Рекомендуется также сохранить граф в файл (File – Save – указать имя и директорию).

Меню Run содержит:

Auto simulation – выполняет моделирования алгоритма от начала и до конца. Созданный отчет дописывается в конец отчета справа. Программа также просит указать номера исходной и конечной вершины.



Step simulation – инициирует пошаговые выполнение алгоритма.

Аналогично предыдущему спрашивает начальную и конечную вершину. Дальше моделирование можно выполнять нажатием Ctrl+N или выбором пункта меню Run – Next step.

При изменении графа рекомендуется инициализировать пошаговое моделирование заново.

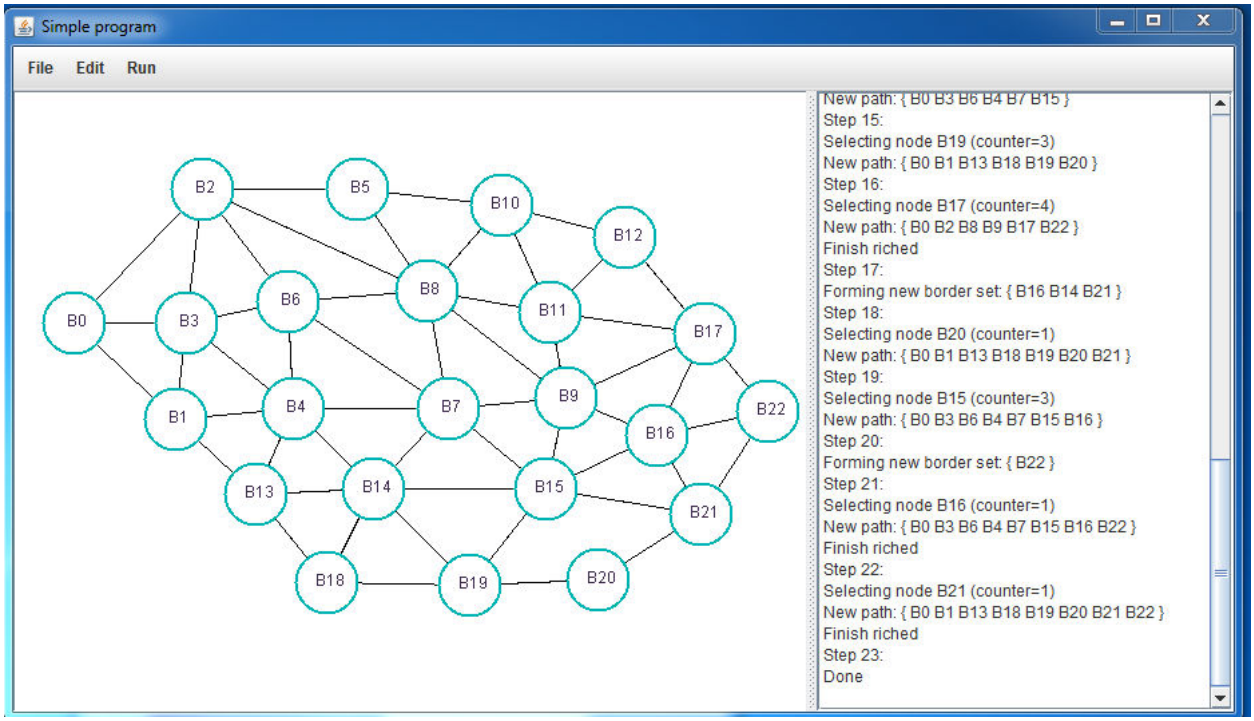
Next step – выполняется один этап выполнения графа. Отчет добавляется в конец отчета справа.

Доступно также по комбинации клавиш Ctrl+N.

Редактор графа

Редактирование выполняется в нескольких режимах (см. описание меню Edit). Основные действие выполняются левой кнопкой мыши, сброс – правой. Откатить действие нельзя.

Пример формирования множества путей от вершины $в_0$ к вершине $в_{22}$



Step 0:

Initial border set: { B3 B1 B2 }

Paths: { B0 B3 } { B0 B1 } { B0 B2 }

Step 1:

Forming new border set: { B4 B5 B8 B6 B13 }

Step 2:

Selecting node B3 (counter=2)

New path: { B0 B3 B6 }

Step 3:

Selecting node B1 (counter=2)

New path: { B0 B1 B13 }

Step 4:

Selecting node B2 (counter=3)

New path: { B0 B2 B8 }

Step 5:

Forming new border set: { B11 B14 B7 B10 B4 B5 B18 B9 }

Step 6:

Selecting node B6 (counter=2)

New path: { B0 B3 B6 B4 }

Step 7:

Selecting node B13 (counter=3)

New path: { B0 B1 B13 B18 }

Step 8:

Selecting node B8 (counter=5)

New path: { B0 B2 B8 B9 }

Step 9:

Forming new border set: { B11 B16 B14 B15 B7 B17 B19 }

Step 10:

Selecting node B4 (counter=2)

New path: { B0 B3 B6 B4 B7 }

Step 11:

Selecting node B18 (counter=2)

New path: { B0 B1 B13 B18 B19 }

Step 12:

Selecting node B9 (counter=5)

New path: { B0 B2 B8 B9 B17 }

Step 13:

Forming new border set: { B11 B16 B14 B15 B20 B22 B12 }

Step 14:

Selecting node B7 (counter=2)

New path: { B0 B3 B6 B4 B7 B15 }

Step 15:

Selecting node B19 (counter=3)

New path: { B0 B1 B13 B18 B19 B20 }

Step 16:

Selecting node B17 (counter=4)

New path: { B0 B2 B8 B9 B17 B22 } сформирован первый путь

Finish riched

Step 17:

Forming new border set: { B16 B14 B21 }

Step 18:

Selecting node B20 (counter=1)

New path: { B0 B1 B13 B18 B19 B20 B21 }

Step 19:

Selecting node B15 (counter=3)

New path: { B0 B3 B6 B4 B7 B15 B16 }

Step 20:

Forming new border set: { B22 }

Step 21:

Selecting node B16 (counter=1)

New path: { B0 B3 B6 B4 B7 B15 B16 B22 } сформирован второй путь

Finish riched

Step 22:

Selecting node B21 (counter=1)

New path: { B0 B1 B13 B18 B19 B20 B21 B22 } сформирован третий путь

Finish riched

Step 23:

Done

Приложение 2. Листинг программы

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package graph;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;

/**
 *
 * @author Holkin
 */
public class Simulation {
    Graph grp;
    Set<Node> border;
    Set<Node> edge;
    Set<Node> closed;
    HashMap<Node,Counter> map;
    Node dstNode;
    ArrayList<ArrayList<Node>> paths;
    public boolean done = false;
    int step = 0;
    public Simulation(Graph g, int src, int dst){
        grp = g;
        dstNode = g.nodes.get(dst);
        Node srcNode = g.nodes.get(src);
        closed = new HashSet<Node>();
        closed.add(srcNode);
        edge = new HashSet<Node>();
        border = new HashSet<Node>();
        border.add(srcNode);
```

```
        paths = new ArrayList<ArrayList<Node>>();
        map = new HashMap<Node,Counter>();
    }
    public String nextStep(){
        if (done)
            return "";
        String ret = "Step "+step++ + ":\n";
        if (edge.isEmpty()){
            ret += "Initial border set: { ";
            for (Node n: border){
                closed.add(n);
                for (Node m: n.neighbours){
                    edge.add(m);
                    closed.add(m);
                    ret += "B"+grp.nodes.indexOf(m)+" ";
                    ArrayList<Node> a = new ArrayList<Node>();
                    a.add(n);
                    a.add(m);
                    paths.add(a);
                }
            }
            border = new HashSet<Node>();
            ret += "}\nPaths: ";
            for (ArrayList<Node> path: paths){
                ret += "{ ";
                for (Node n: path){
                    ret += "B"+grp.nodes.indexOf(n)+" ";
                }
                ret += "} ";
            }
            ret += "\n";
        }else if (border.isEmpty()){
            for (Node n: edge){
                if (n == dstNode)
                    continue;
                for (Node m: n.neighbours){
                    if (!closed.contains(m)){
                        border.add(m);
                        if (map.get(n) == null){
                            map.put(n, new Counter());
                        }else{
                            map.get(n).c++;
                        }
                    }
                }
```

```
        }
        if (border.isEmpty()){
            ret += "Done\n";
            done = true;
            System.out.print(ret);
            return ret;
        }
        ret += "Forming new border set: { ";
        for (Node n: border){
            ret += "B"+grp.nodes.indexOf(n)+" ";
//              if (n != dstNode){
//                  closed.add(n);
//              }
        }
        ret += "}\n";
      }else{
        int min = 10000;
        Node sel = null;
        for (Node n: edge){
            Counter ct = map.get(n);
            if (ct == null)
                continue;
            int k = ct.c;
            if (k < min){
                min = k;
                sel = n;
            }
        }
        if (sel == null || sel == dstNode){
            border = new HashSet<Node>();
            step--;
            return nextStep();
        }
        ret += "Selecting node B"+grp.nodes.indexOf(sel)+" (coun-
ter="+min+")\n";
        closed.add(sel);
        Node nxt = null;
        for (Node n: sel.neighbours){
            if (border.contains(n)){
                nxt = n;
                if (nxt == dstNode)
                    break;
            }
        }
        if (nxt!=null) border.remove(nxt);
```

```
            for (ArrayList<Node> pth : paths){
                if (pth.get(pth.size()-1) == sel){
                    edge.remove(sel);
                    if (nxt == null){
                        ret += "Can't continue path\n";
                        paths.remove(pth);
                        System.out.print(ret);
                        return ret;
                    }
                    edge.add(nxt);
                    pth.add(nxt);
                    if (nxt != dstNode)
                        closed.add(nxt);
                    ret += "New path: { ";
                    for (Node m: pth){
                        ret += "B"+grp.nodes.indexOf(m)+" ";
                    }
                    ret += "}\n";
                    if (nxt == dstNode){
                        ret += "Finish riched\n";
                        paths.remove(pth);
                        border.add(nxt);
                        System.out.print(ret);
                        return ret;
                    }
                    break;
                }
            }
        }
        System.out.print(ret);
        return ret;
    }
    class Counter{
        int c = 1;
    }
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package graph;

import java.io.DataInputStream;
import java.io.DataOutputStream;
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;

/**
 *
 * @author Holkin
 */
public class Graph {
    public ArrayList<Node> nodes = new ArrayList<Node>();
    public void add(Node n){
        nodes.add(n);
    }
    public void remove(Node n){
        nodes.remove(n);
        for (Node a: nodes){
            Node.unlink(a, n);
        }
    }
    public static Graph randomGraph(){
        Graph g = new Graph();
        Random r = new Random();
        for (int i=0; i<r.nextInt(6)+4; i++){
            g.nodes.add(new Node(r.nextInt(500), r.nextInt(500)));
        }
        for (Node n: g.nodes){
            Node.link(g.nodes.get(r.nextInt(g.nodes.size())), n);
            Node.link(g.nodes.get(r.nextInt(g.nodes.size())), n);
        }
        return g;
    }
    public void saveToFile(File file) throws IOException{
        DataOutputStream fw = new DataOutputStream(new FileOutputStream(file));
        fw.writeInt(nodes.size());
        for (Node n: nodes){
            fw.writeInt(n.x);
            fw.writeInt(n.y);
            fw.writeInt(n.neighbours.size());
            for (Node m: n.neighbours){
                fw.writeInt(nodes.indexOf(m));
            }
```

```
        }
        fw.close();
    }
    public static Graph loadFromFile(File file) throws FileNotFoundException, IO-
Exception{
        Graph g = new Graph();
        DataInputStream fr = new DataInputStream(new FileInputStream(file));
        int size = fr.readInt();
        for (int i=0; i<size; i++){
            g.nodes.add(new Node(0, 0));
        }
        for (int i=0; i<size; i++){
            g.nodes.get(i).x = fr.readInt();
            g.nodes.get(i).y = fr.readInt();
            int k = fr.readInt();
            for (int j=0; j<k; j++){
                int id = fr.readInt();
                g.nodes.get(i).neighbours.add(g.nodes.get(id));
            }
        }
        fr.close();
        return g;
    }

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package graph;

import java.util.HashSet;
import java.util.Set;

/**
 *
 * @author Holkin
 */
public class Node {
    public static final int RAD = 24;
    public Set<Node> neighbours = new HashSet<Node>();
    public int x = 0, y = 0;
    public Node(int x, int y){
        this.x = x;
        this.y = y;
    }
```

```
    public static void link(Node a, Node b){
        if (a == b)
            return;
        if (a.neighbours.contains(b))
            return;
        if (b.neighbours.contains(a))
            return;
        a.neighbours.add(b);
        b.neighbours.add(a);
    }
    public static void unlink(Node a, Node b){
        a.neighbours.remove(b);
        b.neighbours.remove(a);
    }
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ui;

import graph.Graph;
import graph.Simulation;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.HeadlessException;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.*;

/**
 *
 * @author Holkin
 */
public class MainWindow extends JFrame{
    public JMenuBar mainMenu;
    public JMenu fileMenu;
    public JMenu editMenu;
    public JMenu simMenu;
    public JMenuItem loadFile;
    public JMenuItem saveFile;
    public JMenuItem newFile;
    public JMenuItem placeNode;
```

```
    public JMenuItem removeNode;
    public JMenuItem moveNode;
    public JMenuItem drawLink;
    public JMenuItem removeLink;
    public JMenuItem stepSim;
    public JMenuItem autoSim;
    public JMenuItem nextStep;
    public Editor editor;
    public JTextPane report;
    public JSplitPane mainPane;
    public Simulation sm;
    public MainWindow(){
        setTitle("Simple program");
        initMenu();
        editor = new Editor();
        EditorListener edlistener = new EditorListener(editor);
        editor.addMouseListener(edlistener);
        editor.addMouseMotionListener(edlistener);
        report = new JTextPane();
        JScrollPane repPane = new JScrollPane(report);
        mainPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, editor, rep-
Pane);
        mainPane.setDividerLocation(500);
        add(mainPane);
        addHandlers();
        setPreferredSize(new Dimension(1000, 700));
        pack();
        setLocationRelativeTo(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    private void initMenu(){
        mainMenu = new JMenuBar();
        fileMenu = new JMenu("File");
        mainMenu.add(fileMenu);
        editMenu = new JMenu("Edit");
        mainMenu.add(editMenu);
        simMenu = new JMenu("Run");
        mainMenu.add(simMenu);
        newFile = new JMenuItem("New");
        fileMenu.add(newFile);
        saveFile = new JMenuItem("Save");
        fileMenu.add(saveFile);
        loadFile = new JMenuItem("Load");
        fileMenu.add(loadFile);
```

```
        placeNode = new JMenuItem("Placing nodes");
        editMenu.add(placeNode);
        moveNode = new JMenuItem("Moving nodes");
        editMenu.add(moveNode);
        removeNode = new JMenuItem("Removing nodes");
        editMenu.add(removeNode);
        drawLink = new JMenuItem("Drawing links");
        editMenu.add(drawLink);
        removeLink = new JMenuItem("Removing links");
        editMenu.add(removeLink);
        autoSim = new JMenuItem("Auto simulation");
        simMenu.add(autoSim);
        stepSim = new JMenuItem("Step simulation");
        simMenu.add(stepSim);
        nextStep = new JMenuItem("Next step");
        simMenu.add(nextStep);
        nextStep.setEnabled(false);
        mainMenu.setLayout(new FlowLayout(FlowLayout.LEFT));
        setJMenuBar(mainMenu);
    }

    private void addHandlers() {
        placeNode.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editor.setMode(Editor.PLACE_NODE_MODE);
            }
        });
        moveNode.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editor.setMode(Editor.MOVE_NODE_MODE);
            }
        });
        removeNode.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editor.setMode(Editor.REMOVE_NODE_MODE);
            }
        });
        drawLink.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editor.setMode(Editor.DRAW_LINK_MODE);
            }
```

```
});
removeLink.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      editor.setMode(Editor.REMOVE_LINK_MODE);
   }
});
newFile.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      editor.grp = new Graph();
      editor.repaint();
   }
});
saveFile.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      JFileChooser fc = new JFileChooser();
      int returnVal = fc.showSaveDialog(mainPane);
      if (returnVal == JFileChooser.APPROVE_OPTION){
         try{
            editor.grp.saveToFile(fc.getSelectedFile());
         }catch(Exception ex){
            report.setText(ex.toString());
         }
      }
   }
});
loadFile.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      JFileChooser fc = new JFileChooser();
      int returnVal = fc.showOpenDialog(mainPane);
      if (returnVal == JFileChooser.APPROVE_OPTION){
         try{
            editor.grp = Graph.loadFromFile(fc.getSelectedFile());
            editor.repaint();
         }catch(Exception ex){
            report.setText(ex.toString());
            ex.printStackTrace();
         }
      }
   }
});
stepSim.addActionListener(new ActionListener() {
```

```
        @Override
        public void actionPerformed(ActionEvent e) {
            simDialog();
        }
    });
    autoSim.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            simDialog();
            String str = "";
            while (!sm.done){
                str += sm.nextStep();
            }
            report.setText(report.getText()+str);
        }
    });
    nextStep.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (sm != null){
                String res = sm.nextStep();
                report.setText(report.getText()+res);
            }
        }
    });
    nextStep.setAccelerator(KeyStroke.getKeyStroke('N',
KeyEvent.CTRL_DOWN_MASK));
  }
  private void simDialog() throws HeadlessException, NumberFormatException {
    JTextField jtf[] = new JTextField[2];
    for (int i=0; i<jtf.length; i++){
        jtf[i] = new JTextField();
    }
    jtf[0].setText("0");
    jtf[1].setText("22");
    final JComponent[] inputs = new JComponent[] {
    new JLabel("From"),
    jtf[0],
    new JLabel("To"),
    jtf[1]
    };
    JOptionPane.showMessageDialog(null, inputs, "Select nodes", JOption-
Pane.PLAIN_MESSAGE);
    int src = Integer.valueOf(jtf[0].getText());
    int dst = Integer.valueOf(jtf[1].getText());
```

```
if (src == dst) {
    report.setText("Destination reached.");
}
sm = new Simulation(ed
```